SOFTBANK MOOK

1999夏号

Oh!X

特集 2D Laboratory

コミック作画はどこまで電子化できるか/フィルタ処理の基本
Photoshopで動くゲーム/無改造MZ-700で画像表示

Welcome to Immersive Space!

華麗なる立体映像の世界/立体視アダプタの製作

Oh!X流デジカメ活用術/Device Programing

SOFT BANK Publishing
オー!エックス
1999夏号
定価2300円

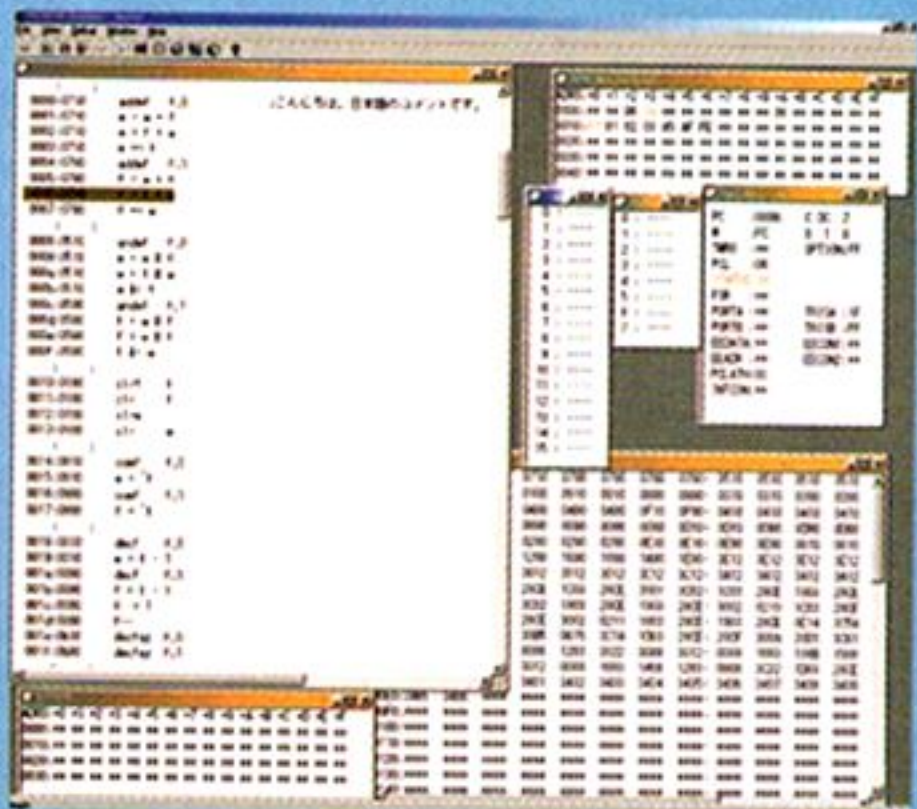
SANDWICH
CAFE CAFE

特集 2D Laboratory

特別企画 Welcome to the Immersive Space

特別企画 Oh!X流デジカメ活用術

Tcl/Tkパズルゲーム

Background Mascot

PICアセンブラ＆シミュレータ

SOFTBANK MOOK

1999 夏号

Oh!X

CONTENTS

●特集 2D Laboratory

●Step to the Black Arts

●Gamer's Eye

●Oh!X68000

©1999 ROMANoV HIGA

E　N　T　S

<スタッフ>
●編集/植木章夫　●編集協力/水上智雄　渡辺真也　●協力/有田隆也　石上達也　市川幹人　桑野雅彦　こて丸　古村聡　清水和人　鈴木典雄　高橋哲史　高橋登志朗　田中順子　都築和彦　中森章　野沢絵美　松尾直樹　広井誠　福原徹　古旗一浩　御木徳高　森川久志　百井洋　吉田賢司　吉田幸一　プロジェクトチームDoGA　満開製作所
●カメラ/蒲生晴夫　●校正/フィールドアップ
●イラスト/岡村直也　高橋哲史　ロマのフ比嘉　福原徹　山田晴久
●編集長/來島樹　●編集人/稲葉俊夫　●発行人/岡崎眞
●DTPデザイン/クニメディア　●印刷/クニメディア

進路

JEFFさん40歳。不景気なんで脱サラで軽食始めたよ。
やっぱ普通じゃ面白くないから空飛んでみたよ。
どーだい兄さんたち。食ってみない？　あ。無視された。
JEFFさん40歳。妻だって説得できたんだ。まだまだ頑張るさ。

表紙　ロマのフ比嘉

沖縄県出身。
第9回DoGA CGAコンテストにてグランプリ獲得。
パトロン計画に参加。「Tech Win」誌でCGアニメーションを不定期連載。最近はDoGAよりビデオを発売。

「ONEDAY、SOMEGIRL」
「BURST」
「SANDSTORM」
「WiredBob」

[BAR SIDE-X]
http://www.pastelnet.or.jp/users/higa/

Xなるものに向かって

シェアウェアというのがどうも好きになれない。Windowsなどでは開発環境自体を揃えるのに出費がかさむので，いく分なりとも回収したいという気持ちもわかるのだが，そもそもそういう気持ちになってしまうような環境というのが好きではない。ことさらにcopyrightをつけるのもなにか鼻についてしまう。昔はそんなの誰も気にしてなかったのに。
周りはまさに鉄の時代。黄金時代からの落差は大きい。オープンソースという潮流も流行り始めているが，開発者自体の数がそれだけ減少しつつあるということの象徴ではないかという気もする。「我が社の製品を皆さんで改良してください」ということの裏返しだ。
Linuxなどはもっと自由に開発されているのかと思ったら，やはりカーネルなどにはあまり手を加えるものではないらしい。一般ユーザーにとっては「無料である」ということだけが大事なのだ。オープンソースだからどうこうと発想する人はまずいない。いろいろ話を聞いてみて私なりに理解したのは，Linuxをはじめとして，いわゆるPC UNIXの目指すところはUNIXであり，それ以外のなにかではないということだ。
UNIXはそれなりに完成された環境であって，裏を返せば進化をやめたOSである。X Windowなどの環境は完備されていてもGUIベースのアプリは非常に少ない。UNIX使いのための環境ではGUIは本質的なものではないのだ。もちろん，GUIであるかどうかというのが問題なのではなくて，UNIXのかたちが決まっているということが不都合なのだ。すべての人がUNIXで幸せになれるわけではないのだから。
一方でLinuxなどを担ぎ出した「オープンソース」という概念はひとり歩きを始めている。マイクロソフトにオープンソース化を要求するなど，「あんなに共産主義の嫌いなアメリカ人がどうして……」と思うくらいソフトウェアの共有に熱心な人もいるようだ。
が，オープンソースを主張するのは勝手だが，他人にまで押しつけてはいけない。Netscapeはともかく，Linuxなんてもともとオープンソースでなかったらド壟壁なOSなのだし，支払う対価が違いすぎる（だから主張してるんだろうけど）。
ということで，「オープンソース」という言葉にはあまりいい印象がない。フリーという言葉さえ存在しないくらいの状態が自然なのだ。GPLを読んで，「なんて堅苦しいんだろう」と思ったことはないだろうか？
X68000というマシンが見せてくれた夢は完成されないままに終わった。結果として，形のない「X」というものだけが残された。「夢」というのは，その「X」の本質のひとつであるから，追い求めてもつかまえられるものではないのかもしれない。が，ひとたび比類なき自由度を知ってしまうと，既存の環境では満足できない。単にフリーウェアが多いというだけではなく，意のままになるツールがほしい。共有の感覚がほしい。もちろん，「すべてのソフトをフリーにすべきだ」などといっているのではなく，黙っていてもそうなる環境を取り戻さなくてはいけないということだ。
なにが障害になるだろう？　最大の問題はユーザーのスキル不足だろう。現状でWindowsアプリなどを公開しても，有効な改良ができる人はどれくらいいるだろうか。公開しても実りある共有でなければ意味がないのだ。だからこそ我々の道は遠く険しい。
ときどき「X」というのはいったいなんなのかと考えることがある。すでにシャープのXシリーズという枠組みで考えてはいない。にもかかわらず，「X」であるべきマシンが存在しなければならない，という思いがある。X68000というマシンはもっとも「X」に近いところにある。が，そのものではない。零式がそうかといわれると，難しい。かなり近いかもしれないし，あるいはそのものかもしれない。が，やはり別のものかもしれない。あるいは参式くらいになるとかなり近いのかもしれないが，その「X」がハードウェアというより，むしろそれを取り巻く環境であるのは間違いないだろう。
我々の多くにとって「X」というのは魂に刻まれた十字架である。一度知ってしまったからには生涯背負い続けるしかないのかもしれない。次の「X」にたどり着くまで。

プロトプラスト

今回紹介するのはチャイナサイバー制作のX68000シリーズ用縦スクロールシューティングゲームだ。CD-ROMにも収録されているので、CD-ROMを接続したX68000ユーザーはぜひプレイしてみてほしい(要ジョイスティック)。

起動すればすぐにわかることだが、大胆にも縦画面オンリーの構成となっている。デフォルトのHorizontalModeでは縦画面を横スクロール(?)風にプレイできる(文字表示などは縦のまま)。Arcadeモードにすればディスプレイを縦置きにしてプレイできる。本来はこちらが正しい遊び方なのだろう。

ステージ構成は2面分と3面のボスのみということで、まだ完成しているものではないが、部分部分の完成度は非常に高いものがある。全体に魔法大作戦というか、その系列のゲームっぽい構成だ。

この手のゲームの例に漏れず、画面内に飛び交う弾数が非常に多いのが特徴だ。たまにスプライトの消し残しが出るものの、雷電2ばりに敵機の破片は飛び散ってもスプライト表示数にはまったく問題がない。

自機は2種類から選択可能。パワー重視タイプとスピード重視タイプだ。さらにそれぞれオプションの展開方法により、前方集中攻撃型、広範囲拡散攻撃型のフォーメーションを取ることができる。これはトリガーを押すたびに切り替わるものだ。押しっぱなしならオート連射するのでトリガーを押し直すことで攻撃形態の切り替えが行われる。連射すれば……集中、拡散を繰り返すだろうが、効率は落ちるだけか?

攻撃武器はショットとミサイル(自動追尾型)で、それぞれアイテムを取ることでパワーアップする。ボムもアイテムで補充可能だ。ショットとミサイルといっても打ち分ける必要はない。ごく普通の地上空中同時攻撃なので、基本的には打ちまくって弾を避けまくっていればいい。

技術的にはほとんど文句をつけようがないだろう。ドットの打ち方も緻密だし、テクノ系サウンドと特異なフォントフェイス、などもいい味を出している。作者の職業、フリーターって、素人さんだよね?

ただ、デフォルトでミサイル装備など、初期装備が結構充実しているためか、パワーアップ感が弱いようにも感じられる。全体的にはやや単調な展開だ(まだ完成品というわけではないのだろうが)。素性はよいのでじっくり仕上げていってほしいと思う。

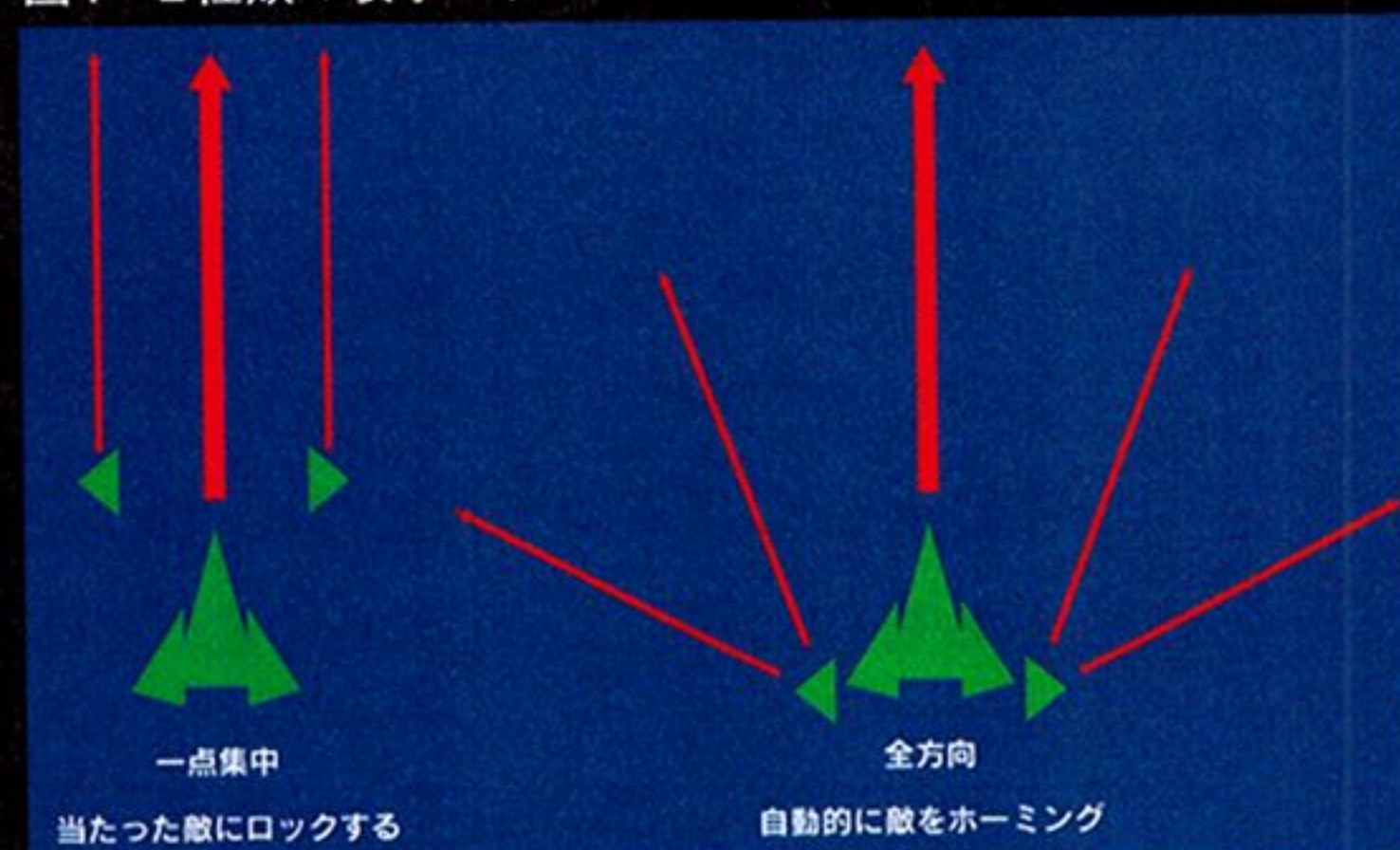
基本的に敵の弾の速度は遅めなので、あわてなければかなり避けられるように設定されている。たまに何種類かの速度の弾が交ざるが、それだけで避け方がぐっと難しくなる。画面内をゆっくり横切る超低速弾もなかなか嫌なものだ。その辺のバランスは悪くない。

ボスの攻撃はそれぞれ数通りのパターンを持っており、単調さはないものの、ぶっといレーザーとかいった特徴は持ち合わせていない。基本的な部分は弾数勝負となっている。ボスなんだから、もう少しくらい理不尽でもいいかもしれない。

ラスボスは死にかけると、それはそれはたくさんの弾を画面中にばらまいてくれる。もう一声エスカレートすると緊迫感がすごくなりそうなのだが。

図1　2種類の攻撃パターン

一点集中
当たった敵にロックする

全方向
自動的に敵をホーミング

Your Memories, My Memories

田中順子 Tanaka Yoriko

一冊のノートから

部屋の掃除をしていたら，なくなっていたものがベッドの隙間や机の引き出しの奥などに隠れていたという経験はないでしょうか。私は，ついこの間そんな経験をしました。ふと，古い本の間から，なくなっていたはずのノートが出てきたのです。

私がパソコンの初心者だった頃，パソコンというものは遥か彼方の世界のものに思えていました。このノートはどうしても覚えることのできない，この異邦人のもてなし方をていねいに書き並べてあるものでした。

もう何年も前になります。CGの魅力に魅せられて，どうしても自分でその作品を創ってみたい衝動にかられていました。未知の異邦人であるパソコンを使う恐怖とCGを創るという未知の世界に踏み込む好奇心との勝負がついたときのことです。いまでも，初めてX68000の起動ボタンを押したときのことが鮮明に記憶に残っています。このときから，このノートとの歩みが始まりました。なにか失敗をするたびにページ数は増えていきました。画面になにも映らなくて，長い間大騒ぎをして，結局原因がモニタのスイッチが入っていなかったこと，コマンドひとつですべてが解決するのに，操作方法がわからず何時間も悩み続けたこと，それから，初めて自分の力でCGを描き上げた記念すべき瞬間。

未知の異邦人から，無二の親友に変わるまでの歩みが，次々と書き加えられていきました。

しかし，月日は流れ，いまではパートナーは違うパソコンに代わってしまいました。そして，このノートには書き込むページはもう残っていません。いまは，新しいノートへ新しい思い出を書き込もうとしています。しかし，これは古いノートを捨て去るのではなく，このノートの上に新しい軌跡を描いていこうとしています。このノートには古い懐かしい記憶と新しい未来への希望がともに見えています。

服装デザインのお話

野沢絵美　Nozawa Emi

こんにちは。前回はネットワークゲームのキャラクターのデザインや世界設定や操作マニュアル書きをやった野沢です。ゲームのドット絵作りにもトライしましたが，結構手こずって普通の原稿を書くよりたくさんの時間と労力をかけてしまったかんじです。ま，ゲーム作りには前から憧れていたので，とりあえず人生の目標をひとつ達成しちゃったってところでしょうか。今回はデフォルトに戻り，エッセイの復活ということで，服装デザインの話でございます。

■女性の服装デザインは苦労する？

［イラストA］

女の子の絵を描くのは大好きだけど，服装デザインは苦手～って方はいませんか？

「資料とか，なにを見たらいいのもよくわからないんですよ～。その昔，『My Birthday』なんかを見て服の研究をしてみましたけどね～」と，服装デザインには悩んでいる人もいるみたいです。それにしても『My Birthday』ってのは占い専門誌だったよーな。まあ，考えてみると『男がひとりで女性誌を買う』というのは，かなり勇気がいることかもしれません。

確かに男がひとりで女性誌コーナーへ行き，立ち読みをしている女性の間に入り，「今年の夏の流行ワンピを1万円以下で買っちゃおう！」「だれでもできるらくらくダイエット」「彼氏に気に入ってもらえる小顔コスメ大研究」なんてタイトルの雑誌を手にし，ささっと内容を確認し，レジまで持っていく……，って光景を見たことはありません。私なんか，男が女性誌コーナーなんかをうろちょろしていたら「こいつ痴漢じゃないか～」って疑っちゃうもんね。確かに女性誌コーナーは男が近寄れるスキってものがない。

それなら店頭でじかに服を見るのはどうかと思うと，これはさらに難易度の高い手段と気づきました。なぜって，女性服の売り場はさらに男性密度が低いんですね～。男性は女性同伴じゃないと店内はうろつけません。ひと昔前のディスコと同じです。バーゲンといった混雑時は女の戦場と化して，気弱な女性は突入を躊躇するほどです(突き飛ばされるよ，マジで)。もちろん，連れの男なんざぁ入るスキもありません。知人の男性は「彼女でも一緒にいないと女性服の売り場を歩くのは目立つからつらいし，店員が『なにかお手伝いしましょうか？』って寄ってきたときにゃあ逃げちゃいますよ～」とコメントしました。

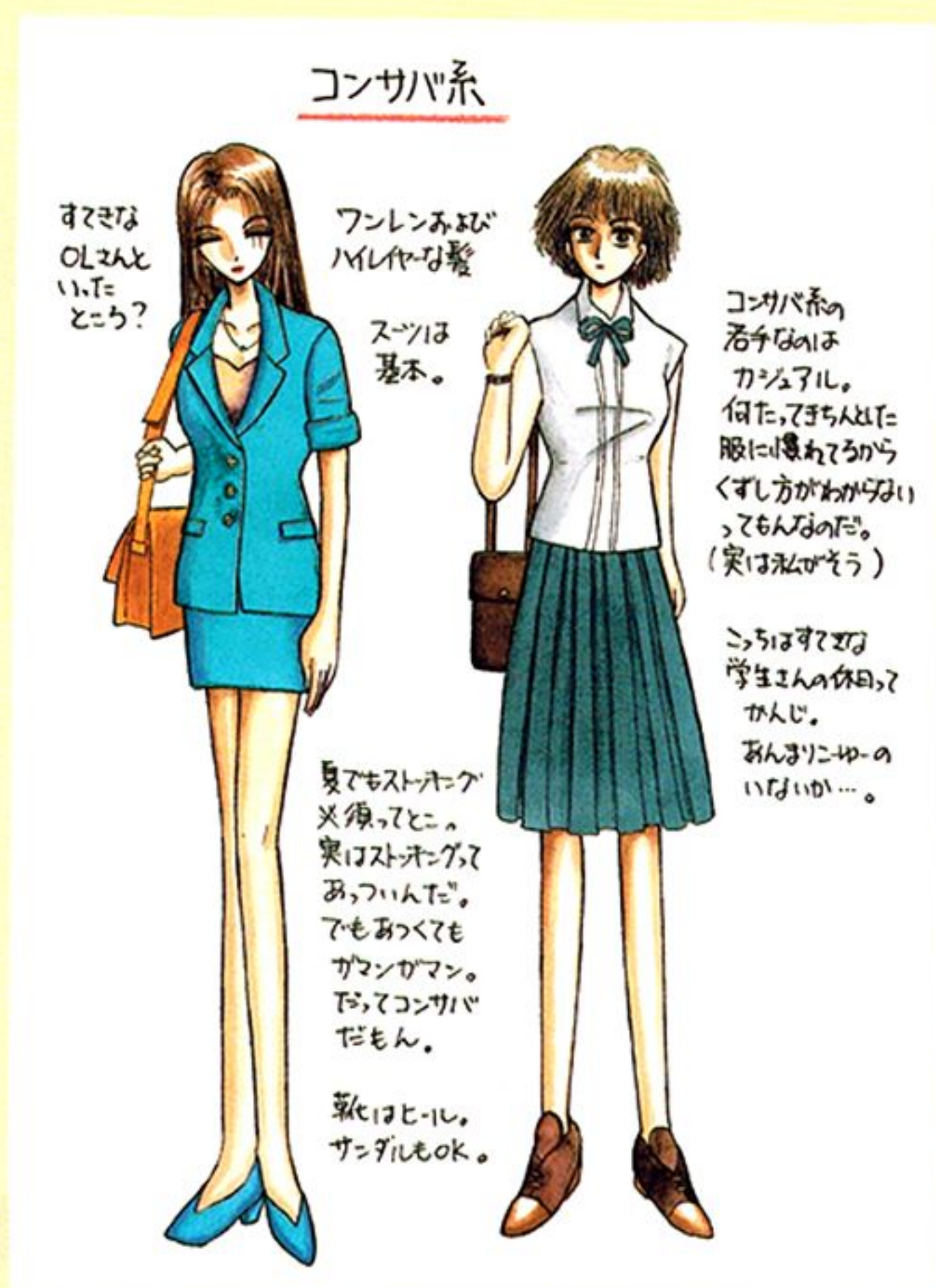
うーむ，男社会に女が進出してきた時代となり，コップ酒が似合うモツ煮込み専門店とか競馬場にギャルが出現し野郎の居場所が侵食されつつあるのに，華やかなシーンに野郎どもが顔出しできる時代はまだまだってな雰囲気です。殿方からそんな現状を告げられて，自分で注意して見るまで気づきませんでした。男はなんだかんだで大変だ～。

実は私だって服装デザインを楽勝でやっているワケではありません。だって，女性の服ってのはいっぱいいっぱい種類があるし，流行だって毎年あれやこれやと登場するからです。キャラクターの服をひとつ決めるのにもかなりの時間がかかります。結構大変なんです。それでも，この苦しい過程が好きだったりもします。大変ではあるけど楽しい作業なんですよね。ということで今回は私が女の子の服装デザインをするときのお話をしてみましょう。

■キャラクターの性格や立場を把握する

キャラクターの服装デザインをするときには，まずそのキャラクターの性格や立場を大まかに決めるようにします。服というのはその人の性格，地位，趣味，人柄，購買力ってのが表れるものだからです。たとえば，性格が明るい子は明るい色使いや綺麗な模様の服を好み，自分のかわいい部分をアピールしたい子はフリルとかリボンのついた服やキティちゃんとかテディベアなどかわいいグッズを身につけ，男勝りで自分の女性的な部分を認めたくないボーイッシュな子は行動的なパンツルック，セクシー度で自分の存在をアピールしたがる人は露出度が多かったり派手な色使いの服を好み，自分の趣味や仕事に夢中（アマチュアやプロ）でほかのことに頭が回らない人は服装に無頓着などなど。

人が服を買うときはたいていはお気に入りのブランド（あるいはスーパー）があり，そのお店で購入します。お気に入りのブランドがなくても同じ傾向の服に手を出すものです。そして心境の変化があると服の好みも変わってきたりします。そういうものなのです。だから，最初にキャラクターを知っておくと，服の好みも絞れてくるので，膨大な種類がある服の中からあれこれ選ぶ手間も少なくなり，デザインがしやすくなるのです。

イラストA

イラストB

購買力だって服に表れます。女子中高生の目立ちたい盛りであり，流行を追いかけるのが大好きで，気合を入れておしゃれをしている今風の女の子はスカート，ワンピース，キャミソ，カーディガン，靴，アクセサリとあれこれいーっぱいほしいものがあります。でも購買力には限界があるので価格的に高価なものは身につけられない。となると，見た目を優先させ，縫製や素材で価格を抑えている服になります。1着2,000～5,000円クラスの服。こういうのは，素材が化繊で，ボタンはプラスチック，裏地はナシまたはついていたとしても薄いって感じです。

大学生やOLになると購買力がついてきているので，それなりにきちんとした縫製のブランド服とか，カシミヤやシルクといった質のいい素材の服になってきます。1着5,000～5万円クラスってところでしょうか，安っぽくは見えません。

さらに服に金をかけられるようになると，マックスマーラーとかフェンディといった有名デザイナーズブランドが登場します。ここあたりになるとちょっとした服でも10万円以上になってきます。デザイン，染色技術，縫製，素材はいいのが当たり前，さらにブランドのロゴマークが目立つようになります。とどめのシャネルとかクリスチャン・ディオールとかラクロワとかはカーディガンが20万円，スーツは50万円ってところです。これになると普通のOLとか学生じゃ太刀打ちできません。特殊な立場の人しか買いません（買えません）。

どこが普通のブランド服と違うのかといわれると，実際買ったことがない私では詳しく説明できないのですが，デザイン，縫製，布，染色，加工，アクセサリにさまざまな人件費と技術料が組み込まれているからじゃないかと推測できます。実際，一流のデザイナーズブランドの服は，布地や，レースもオーダーメイドです。裏地だってそのメーカーのロゴが入っている専用のものです。しっかりした作りのうえに，レースなどの加工が綺麗に施されているので，遠くから見てもゴージャスかつエレガントです。

こんな感じで，服装が着ている人を表しますし，人だって初対面の相手は服装を見て「年は○○歳くらいで，職業は○○で，性格は○○ってかんじの人かな～」と判断するものです。

なので，服装デザインをするときにはそれを逆手に取ります。ゲームやマンガのキャラクターはそのキャラがどういった人物なのかを文章で表すのには限界があります。特にポスターやパッケージ絵や挿し絵などには文字がほとんど使えないからです。そうなるとキャラを説明する手段は服装と小物と背景のみ。ユーザーさんにそのキャラクターがどんな人物なのかを予測してもらうために，こっちから服装で理解してもらおうと仕掛けるのです。キャラクターをわかりやすくユーザーに説明する作業ともいえます。

もちろん，枠にはまらない例外だってあるでしょう。しかし例外はあくまで例外であって，登場人物のほとんどが例外だとユーザーだって混乱しちゃうってもんです。

そういったわけで，最初にそのキャラクターを知る必要があるのです。キャラクターがわかってくるとその子が着たがっている服装の傾向も絞れてくるので，デザインがしやすくなります。

■服装系統みたいなもの

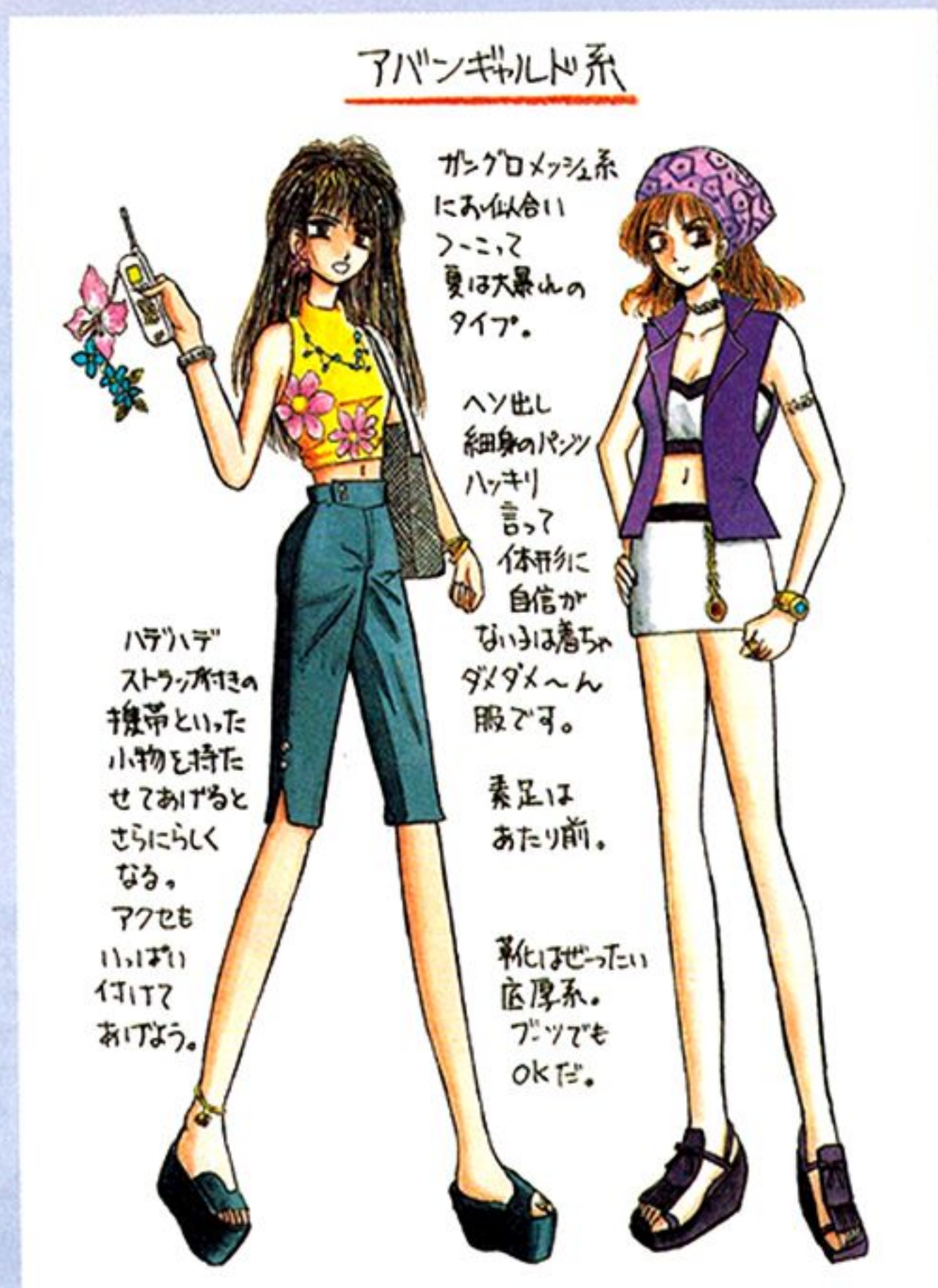
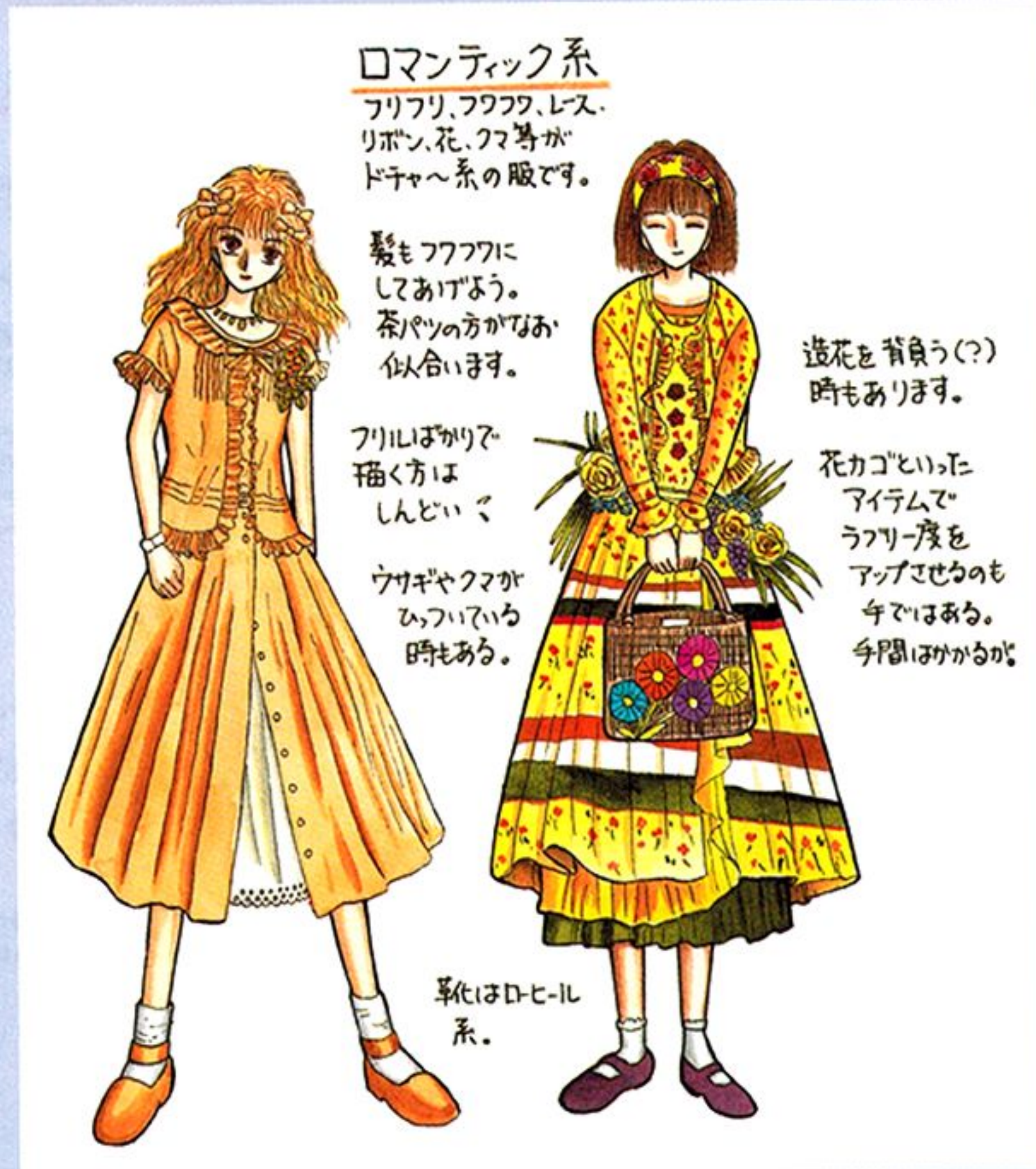
性格が決まっても，その性格にはどんな服装にしたらいいかわからないという方向けに，とりあえず服装の系統分けをしてみました。しかし，これは私が勝手にやった大ざっぱなものですので，これがファッション界の系統分けだと誤解しないでくださいね。あと，専門家の方もつっこみはナシでございます。

・コンサバ系［イラストB］

『美しい，上品，社会人』がキーワードとでもいえる服装を指します。質のいいスーツ，ワンピース，シンプルなブラウスとスカートの組み合わせなどエレガントな雰囲気を醸し出す路線です。対応人種はOL，お嬢様，有閑マダムってところでしょうか。靴もヒール系がメインです。それも，1万円以上はするきちんと作られた靴です。スカーフを上品にアレンジして身につけるのもコンサバ系の特徴です。一流メーカー製の時計，本物の石や貴金属を使ったアクセサリ，革製のバッグと小物もきちんとしたものがメイン。過度な装飾やポップなアイテムはこの系統には当てはまりません。あくまでエレガントで攻めていきます。

コンサバを目指すにはお金がかかりそうですが，1つひとつの持ちものが流行に流されず，変な自己主張をしなくて，質がいいものなので長持ちすることもあり，長い目で見る人にはそれほど険しい道ではありません。

・アバンギャルド系［イラストC］

ミニスカ，カプリパンツ，ミニのキュロット，ショートパンツ，巻きスカート（パレオ系），透けるキャミソ，ノースリーブといったセクシー系の服装です。色づかいは明るめが多いけど……それだけとはいい切れません。パステルカラーから

イラストC

イラストD

ビビッドな原色，黒やデニムの青までとさまざまです。模様は花柄とか豹柄，小物は，布製のバッグや，プラスチックやガラス製のアクセサリー，キティちゃんなどと，かわいくて，綺麗でそんでもってそこそこ安いものがメイン。

しかし，ときたまヴィトンやプラダやシャネルのバッグが出てくるので油断大敵です。靴は身長が高く見える底高靴，サボ，ブーツやカラフルなスニーカーなどが好まれてます。かっこいい，目立つならなんでもOKなのです。自分の女らしさをアピールする服装がアバンギャルド系です。購入価格も安いのから高いのまでいろいろです。自由度はいちばんですが，綺麗じゃないとダメーんってところです。対応人種は，コギャルからOLまででしょう。それ以上はちょっとつらいってところです。原宿，六本木，渋谷あたりで大量にサンプルを発見できます。

・ロマンティック系［イラストD］

大草原の背景がよく似合っているフリルとリボンとレースがメインの服装を好む派です。スカート丈は膝下とかロングスがほとんどでミニはお目にかかりません。色はパステルカラーや原色やモノクロとさまざまなのですが，花柄や熊の模様といったプリント柄がたくさんあるってのも特徴です。王子様を夢見るかわいい性格や，草原で花を摘むってなかんじが似合う少女向きの服装です。甘いものが好きだったり，自らもお菓子作りが好きな子が多いのもこの系統。熱心なキティちゃんファンもおります。学生さんから主婦，おばあさままで幅広いファンがいるので対応人種が幅広い感じです。全体的におとなしい性格の人が好む服装です。

話はずれますが，ロマンティック系大代表のピンクハウスはコミケ会場でもよーく見かけます。コミケ会場にはピンクハウスの同人誌まであるそーです（買ったことないけど）。ってなことでアニメやゲーム方面に夢見る少女にもファンは多いんでしょうかね～。

・ナチュラル系［イラストE］

綿，麻，シルクといった自然の素材にこだわり，色はモノトーンやモスグリーン，ワインレッドといった穏やかな色合い，草木染めってのもあります。デザインは体のラインを強調せず，ゆったりした穏やかなロングスカート，ロングカーディガンなどシンプルなデザインのもの。ときたまシンプルとは反対に「ファンタジーの登場人物が着るのかいな？」って感じの宗教がかったむちゃくちゃなデザインもあります。

とにかく素材重視なので，デザインがシンプルでも高い服が多くなります。ナチュラル系は化粧だって自然「すっぴん」に見えるように仕上げます。対応人種は，性格的にきちっとした自分の世界を持っている人でしょうか。作家とかクリエーター系が多い傾向にあります。いい素材を使うため，地味な服のワリには価格は高いです。そこそこ収入がある人でないと買えません。下手にナチュラル系をデザインすると単なる地味になってしまうのが悲しいところです。デザインやイラストの腕が出る手強い系統です。

・激ハデ系［イラストF］

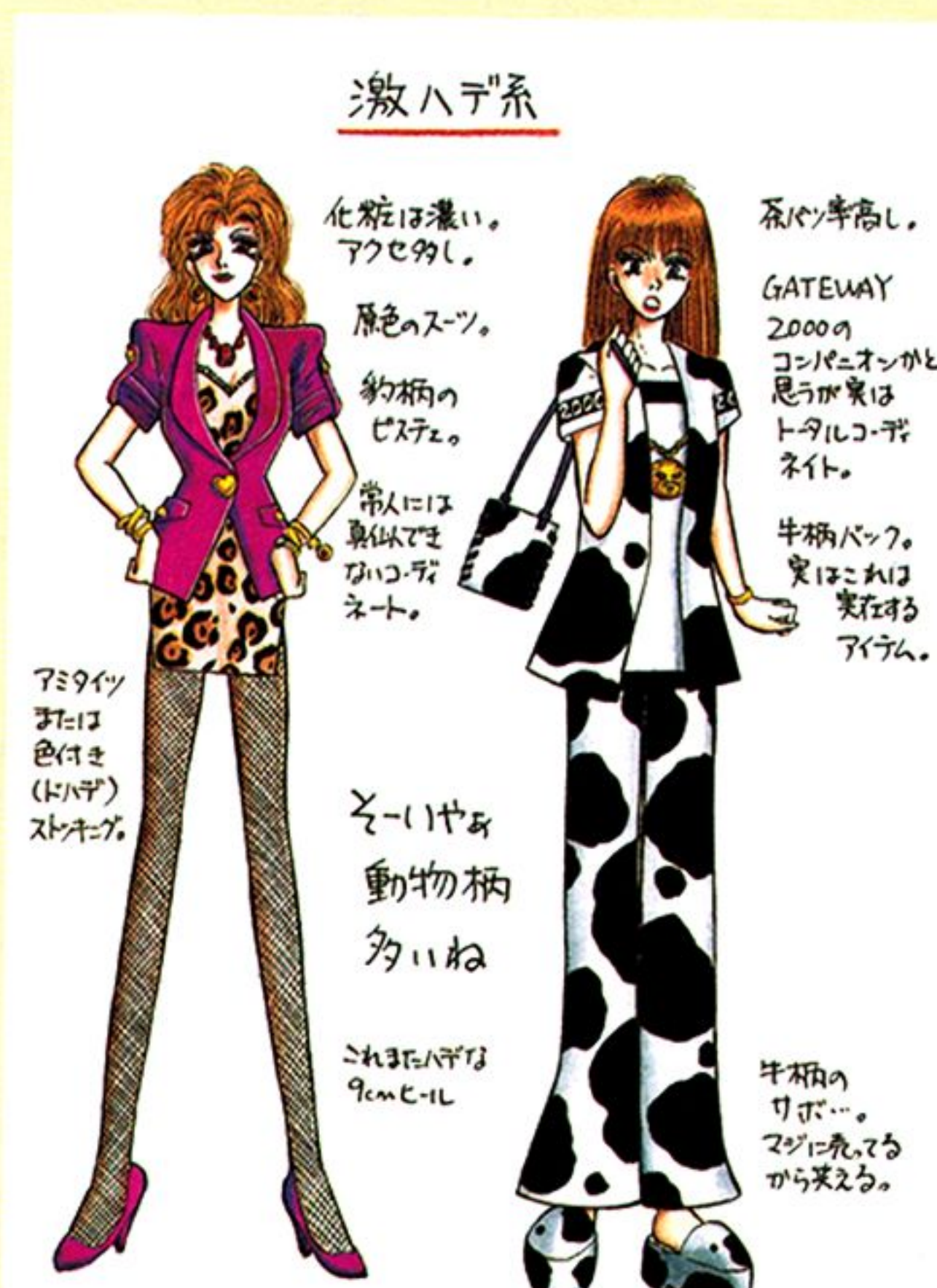
とにかく目立ちたい一心で，節度なく目立ち，かつ下品な感じを醸し出すタイプです。アバンギャルドやコンサバが間違えるとこっちになります。普通の感性や常識がある人なら着ないデザインや色づかい，出しすぎた肌，凝りすぎて浮いちゃっているデザイン，華美すぎるアクセサリ，成金趣味の宝石，服にあっていない靴などなど。見た目重視のハデな服や靴ってのは意外と安いものなので，お金はそんなにかかってません。大きな貴石（アメジスト，トパーズ，アクアマリン）もけっこう安いものなのです。まあ，綺麗に目立ちたいのなら，おしゃれにそれなりのお金をかけないとダメってことですね。対応人種はお水系，または自分が周りにどう思われているか無頓着な人，成金おばさんなど。若い時代ならパワーと愛嬌で押し切れますが，年が進むにつれて見ているほうがつらくなってきます。んでも，こういったキャラクターも作っていると楽しいもんです。

・グランジ系［イラストG］

ヒッピーみたいなハギレをつなぎ合わせたきっちゃないかんじのファッションをグランジと呼んでます。一流デザイナーがグランジを発表したりはしていますが，あまり一般社会で着ている方を見たことはありません。わざと汚くしているんですからね～。お金払って汚くなる服装ってのはふつー着ないでしょ。安全靴，ブーツがぴったり合う服です。対応人種は，も，若い子でしょ。おばはんでこれをやってたらちょっと怖いですし。「熊さんぬいぐるみより拳銃が好き」「あたしは普通の女の子ではない」と主張する個性派の女の子に使えそうなので，想像の中のキャラには効果を発揮しそうです。

・カジュアル系［イラストH］

男の子っぽかったり行動的な人向けの服です。

イラストE

イラストF

イラストG

スカート，キュロット，パンツ，Tシャツ，カットソー，パーカー，Gジャン，スニーカーなど動きやすい服装です。模様は水玉やストライプといったシンプルなものからユニークなものまでさまざま。カジュアルはシンプルさゆえに，かわゆくなるか，ダサダサになるかの紙一重的緊張感があります。微妙な服の長さやデザインなどで，体のラインが綺麗に出たり反対に太って見えたりするからです。素材だって若いうちは安めの素材でも許されますが，社会人となるとある程度，質のいいものでないと寂しい感じになります。大人の女にとってカジュアルってのはコンサバより難しいのです。対応人種はとりあえず全般。誰だってカジュアルな服を一着は持っているはずです。学生には定番の服装ではないでしょうか。自由業の人もカジュアルを好む傾向にあります。

・スーパー系［イラストI］

デザインがシンプルなセーター，シャツ，膝下スカート，ウエスト部分がゴムのパンツと，横幅が3Eの靴，スーパーで手軽に見つけられる系統です。服は無難なデザインで誰にでも着れるものというのが特徴です。しかし誰にでも着れるってのがくせ者で，なかなかバシッと決まらない服でもあります。あまり目立ったりもしません。対応人種は子供からおばあさんまで幅広いです。多分，女の子の服が苦手な方でも普通系の服は描けるでしょう。でも，普通系は見た目が勝負の2Dや3D系キャラクターには不向きな服といえます。

と，だいたいこんな感じに大ざっぱに分類して，キャラクターの服装をデザインます。もちろん人にはいろいろなタイプがありますから単純に系統分けできないケースだってありますし，服だっていろんな種類があります。さらに制服とか和服なんかはどーなるのかい？　ってこともありますが，とりあえず，「若い娘さんとOLあたりが対象の服を大まかに分類してみましたぁ」とのことでご理解くださいね。

●複合技［イラストJ］

さて，性格や立場に基づいて洋服をデザインするといいましたが，これにこだわってあまりに厳密にやってしまうと，一辺倒のデザインになってしまいますし，見るほうもデザインするほうも飽きてしまいます。それにキャラクターだってそんな単純な性格ばかりではありません。心境の変化や流行によって買う服の種類が変わるときだってあるもんです。

そこで，ロマンティックなのとアバンギャルドな服装を合わせたりするなど，違う系統で組み合わせをしてみます。意外なデザインができあがってかえって新鮮なこともあります。たとえば，ロマン系のフリルいっぱいスカートの上にGジャン羽織らせたりするなどです。実際の着こなしバリエーションとしてもこういった方法が紹介されてます。系統が混在した服を取り扱っているデザイナーズブランドだってあります。

ただし，複合技はあまりしつこくやると，キャラクターの特徴が見えなくなってきたり，服自体が複雑怪奇になるので，あまり多用しないほうが無難です。ときたま，原宿とかモード関係の専門学校近辺で複合技失敗例を見かけます。そんなときはマジでビックリ仰天します。オリジナルのかっこいい服装をデザインしているつもりで「なんかうまくいかないな〜」って思ったときは，系統をめちゃくちゃに組み合わせてないかチェックしてみましょう。凝りすぎも失敗のもとです。

●色と模様［イラストK］

色を変えることで同じデザインの服の印象ががらりと変わります。赤や濃いピンクは元気といったイメージがあるので，行動的な女の子の服に使うと色だけでも性格が伝わりやすくなります。また，青や濃紺は冷静や知的なキャラクター，パステルカラー（ピンクや薄い黄色，緑，水色）は若々しいとか愛らしいイメージ，黒や濃い紫はなにかを秘めている危険な雰囲気などです。そんなに簡単に色で決められるもんなのかと思いますが，けっこう色というのは人に影響を与えるものなのです。

会社の仕事である広告やパッケージの作成時に，色には注意しろと教えられました。色によってパッケージの売れ行きに差が出るからです。たとえば赤。赤は「一番を取りたい」ときに使うと効果的な色とのこと。赤をどーんと使ったロゴ，広告，パッケージ，製品なんかはヒットを出しやすいんだそうです。赤の戦略商品の例で有名なのがコカコーラ。あの真っ赤なロゴは赤の効果を狙って作られたものだそうです。確かに街を歩いていても視覚に飛び込む，記憶に残りやすい，なにか惹かれるものがあります。

アニメだと魔女っこもののヒロインの洋服が赤，特撮だと戦隊もののリーダーは赤。色には，色が出す波長というものが存在し，赤はその波長が一番長い色なのです。人は無意識に目に飛び込んだ色の波長に心を左右されてしまうのではないでしょうか。こう考えると赤の戦略というのはあながち伝説ではないみたいです。

カジュアル系
何だかんだでカジュアルが一番気楽に着れてみんな持ってるけど，着こなすとなると努力とセンスが必要…だと思う。
Tシャツの重ね着。今年はこれがうじゃ〜と出てますね。
LOVE
LOVE
LOVE!
フリルスカートはひざギリギリがライン的にもきれい。
靴はローヒールかスニーカー。

イラストH

スーパー系
これはやめましょう。
HELLO WORLD
おしりのラインで切れるTシャツ。実は一番太い所が強調されるので，太く見えてしまうのだ。
ふくらはぎのあたりで切れているスカート。これも一番太い所が強調されてしまうので足も太く見えてしまう。
2,980円のパンプス。それもEEE。足には健康的でいいけど，ヒロインにゃやめた方がいい。

イラストI

複合技
実はBATUってブランドがこの系統。あたしも一着ほし〜!!
Gジャンとフリフリスカートの複合。一歩はずすとダサダサになる。
カジュアル
＋
ロマンテック

イラストJ

だから，色といえども，キャラクターの性格に合わせ，ユーザーさんにそのキャラクターにどういった印象を持ってもらいたいか，考えて使えば効果的でしょう。色についての専門書なんかを読むとさらに参考になります。

模様は，ストライプ，チェック，花柄，アニマル柄などなどありますが，あまり細かい模様をちまちま描くのは避けています。細かいと，面倒くさいですし，そんなに目立った効果が感じられないからです。服全体といったあまり広い面積に模様を施すと，装飾華美にもなってしまいます。なので，袖，襟，スカートに部分的に模様を施します。模様の応用として，どんな服を着ていてもキャラクターのお気に入りの模様（例：花や紋章など）がさりげなく施されているって演出もあります。この模様とキャラクターとの関係とは……って想像しながらデザインを作成していくと，こっちもなんだかワクワクしてきちゃいます。

●小物［イラストL］

服装が決まったら靴，アクセサリ，バッグ，手袋，帽子といった小物を用意します。小物といえどもキャラクターの個性や趣味が表れるアイテムです。きちんとデザインしてあげれば，キャラクターが光ってくるので，これを使わないのってのはもったいない気がします。

私は必ずなんらかのこだわり小物をつけるようにしています。小物は服装に合ったデザイン，色，素材，雰囲気のものを身につけるのが基本ですが，これはそれほど厳密というわけではありません。かちっとしたスーツにスニーカーといったミスマッチな組み合わせがかっこいい場合もあるからです。キャラクターの個性によってはきちんとしたのがいい人と，ミスマッチが似合う人に分かれるとは思います。ここは，キャラクターとTPOに合わせて組み合わせてあげるのがいちばんでしょう。

小物のなかでも絶対必要なのが靴です。靴は大まかにハイヒール，ローヒール，サンダル，サボ（ぽっくりみたいなやつ），ブーツ，スニーカーの感じで分類しています。靴も系統によって好みが分かれますし，キャラクターの置かれている状況や季節によっても靴の選択が左右されます。たとえばハイヒールは都会向けですし，山岳地帯だと登山靴（ブーツ系），浜辺だとサンダルがグゥといったぐあいです。そして，季節によっても靴の向き不向きがあります。

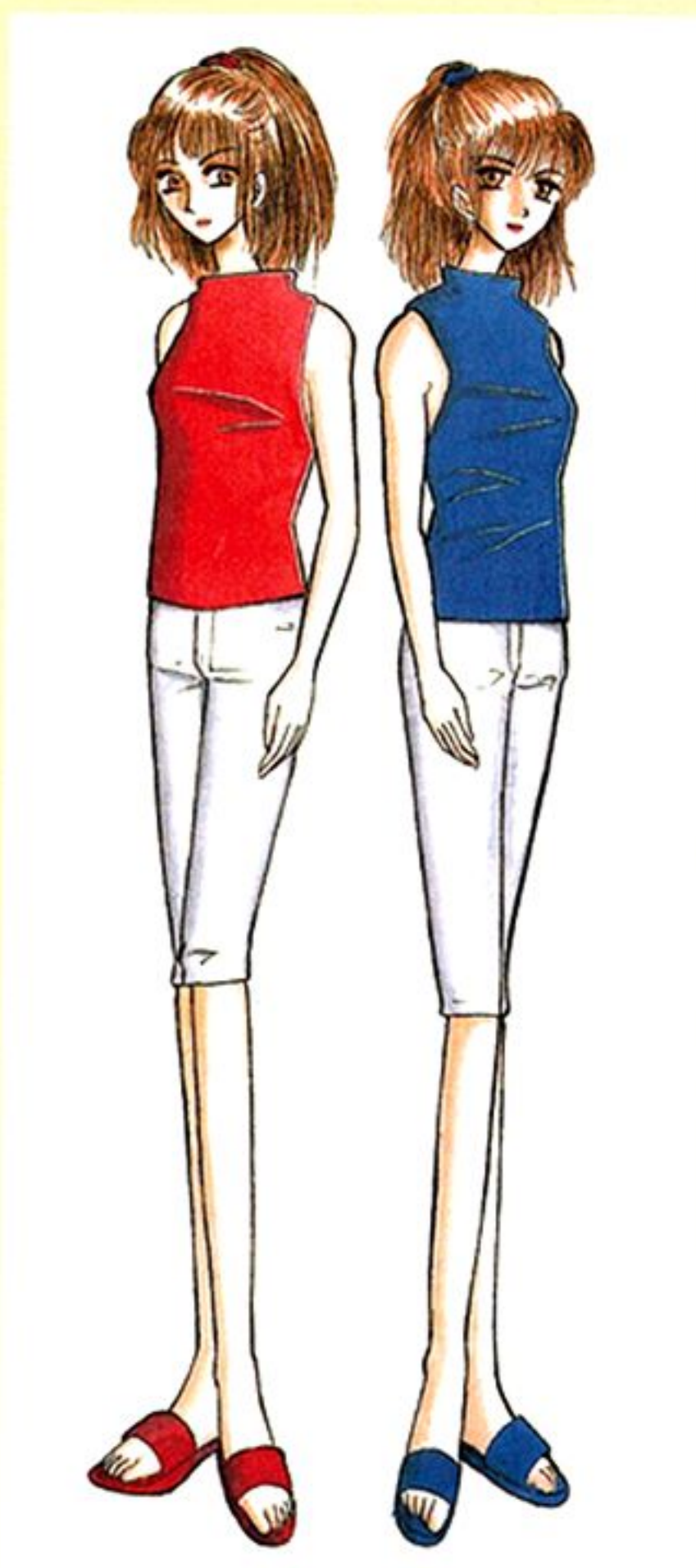
まあ，季節感による靴の変化ってのはここ数年どちゃーって崩れ，夏でもブーツ履いている少女がいるので，あくまで目安ですが。あとはキャラクターと相談してその人の好きな靴を履かせてあげます。アクションキャラなのにスーツとハイヒールが好きというのなら，ハイヒールのお姿でバシバシアクションをさせてあげます。普通の女なら捻挫もんですが，仮想の世界はこんなのも許されるってもんです。なにより，ハイヒールで走れる女ってのはそれだけで驚異的であり超人的な存在ですしね。

ネックレス，ブレスレット，イヤリング，ブローチ，髪飾りといったアクセサリは，ちょっと服装が寂しいなってときにつけてあげます。あまりつけすぎても下品になるので，1～2個くらいをつけると落ち着きます。もちろん下品系にしたいときは大量にアクセサリをつけてあげると，手軽に下品になってくれます。アクセサリの素材は，金属，貴石，パール，プラスチック，ウッド，布，ビーズ，ガラスといろいろ。夏はガラスやビーズ，シルバーといった涼しげな素材，冬はゴールドや木，布といった温かみを感じさせるものが好まれます。季節によってつけられるものが左右されるアイテムでもあります。「このネックレスはキャラクターの大切な思い出の品にしちゃった」といった複線を設定しちゃうこともあります。こんな煩悩が浮かんでくるとアクセサリデザインってのも楽しくなります。

●資料［イラストM］

きちっと状況設定をして服装デザインをするといっても資料があるのとないのとでは大違いです。私は資料として主に雑誌やムック本を使います。また，テレビに出てくる女優さんの服をチェックしたり，街で女の子をリサーチしまくったり，会社の同僚が可愛い服を着ていたら，すかさずブランド名を聞くなど，日々の生活のなかで情報収集をしています。職業意識がすごいってのではありません，自分のほしい服のリサーチを兼ねちゃってるのです。のほほほほ。

・雑誌（andムック本）

イマ風の女の子用の服装をデザインするのに最適な資料として雑誌を見るようにしています。流行をいち早く押さえ，特集として掲載してくれますし，写真もプロが撮っただけあって服の細部も分かりやすく綺麗です。服だけではなく，靴やアクセサリーといった小物も同様に載っているので手っ取り早くあれこれの情報を入手できます。さ

イラストK

イラストL

イラストM

らに服を組み合わせることでさまざまなバリエーションを生み出す着こなし技なども紹介されるので，デザインのネタにつまったときには助かったりします。雑誌に掲載されているデザイナーズブランドの広告写真はおしゃれだし，モデルさんは綺麗なので，イラストを描くときの参考になります。こういった雑誌は読者の対象年齢層があり，その人たちに向いた価格帯の服装を紹介してます。

服装デザインをしたいキャラクターにあわせて，コギャル向けの雑誌かOL向けの雑誌かを選ぶと希望どおりの服装デザインをゲットしやすいです。

イブニングドレスや高級なスーツのデザインなんかがほしいときは，パリまたはミラノコレクションなど海外の一流デザイナーズブランド特集が載っているムック本からネタを仕入れます。さすが一流デザイナーがデザインしただけあってデザインもさまざまですし，色づかいも極まってます。毛皮，シルク，麻といった素材を上手に組み合わせているので，こっちも勉強になります。とにかくぶっ飛んだゴージャスなデザインがほしいときは『オートクチュール*1特集号』，高級なスーツなどのデザインがほしいときは『プレタポルテ*2特集号』を選ぶようにしています。

女性誌がほしくても買いにくいという男性は，外国の専門雑誌を狙ったらどうでしょうか。日本語版より価格が高いのですが，大きい書店なら，ELLE，COSMOPOLITAN，VOGUE，Marine Claireといったファッション雑誌を置いてます。外国書籍コーナーは人も少ないし，なんとなく英字がかっこいいので男性でも比較的買いやすいのではないでしょうか。(ダメ？)

*1 オーダーメイドの服のこと
*2 つるしの服のこと

・カタログ

意外ですが，デザイナーズブランドはカタログがあまり用意されてません。ブティックが店頭で自社ブランドをお客様に見せるときは，雑誌に掲載になった記事の切り抜きファイルを見せたりしてます。場所によっては雑誌がそのまま置かれていたりもします。あのシャネルでさえカタログは販促として購入客にプレゼントする程度しか用意されてませんし，そのカタログもイメージ写真集なのでデザインの詳細がわかるものではありません。シャネルも購入相談時用としてショウの生写真とか雑誌の切り抜きを用意しています。

あたしゃあ何回もシャネルに足を運びましたが，新作デザインが勢揃いっていうカタログなんざぁにはお目にかかったことがありません。カタログを用意しているところもありますが，購入客に配るところがほとんどです。お得意様に「また来（着）てね」とのことで手渡したり郵送したりします。つまり，一度は買わないとカタログは入手しづらいのです。まずはカタログを集めてから……ってなパソコン野郎の掟が通用しない業界のようです。

・テレビ

ファッション専門番組を見るのも大好きです。一流のデザイナーズブランドを取り上げるのがほとんどですが，モデルさんの後ろ姿も見れるので，普段雑誌でお目にかかれない部分のデザインも見れます。ひとつのデザインが映るのはだいたい2～3秒，長くて5秒ってところなので，ちゃんと研究したいのならビデオに撮って見るのがいちばんです。

・インターネット

パソコン野郎のお得意情報収集方法でもあるインターネットでも服の情報は入手できます。しかし雑誌ほど掲載されている作品数が多くなかったり，ブラウザ上で表示できる程度の写真だと細部がわからなかったりもします。ぼやぼや～とした写真を眺めてその細部が想像つく強者ならインターネットの写真からデザインを起こせるでしょうが，初心者にはちょっとつらいかもしれません。

それに，どうもインターネットには不慣れなデザイナーズブランドが多くて，ホームページの情報が希薄だったり，デザイン製を追求するあまりめちゃくちゃ重かったりと，とんでもないホームページもあります。どっちかというとELLEといったファッション専門雑誌が運営するホームページのほうが情報が新しいし，綺麗な写真が載ってます。私的にインターネットはそれほどおすすめではないのですが，手軽に自宅から検索できる点ではひとつの手段かもしれません。

*ココはOKのホームページ
『ピンクハウス』
http://www.pink-house.com/
ロマンティック系のフリルたくさん服をデザインする人は必ず押さえたいブランド「ピンクハウス」の衣装が掲載されているホームページ。レイアウトもホームページ軽さも及第点です。コレクションの写真やムービーコーナーもあるのでグゥ。

・通信販売のカタログ

最近少女画に目覚めた友達のひとりは，通販カタログを愛用しているそうです。通販カタログはデザイナーズブランド等は取り扱ってませんが，ひととおりのファッションが揃っています。さらに，下着やストッキング，靴の写真まであるので，これは意外に役立つ資料です。彼の着眼点に感心しちゃいました。通販カタログに載っている服は無難なデザインが多いので，スーパー系になりがちですが，まったく資料が手に入らないのならこれも手のうちかと思います。

●自分なりのアレンジを加える

さて，キャラクターのアピール方法も決まり，服装も決まり，それに似合った小物も選んだら，あとはデザインをするだけです。資料からキャラクターに似合う服を探し，そっくりそのままデザインするのも悪くはありませんが，それでは物足りないってもんです。

せっかくオリジナルのキャラクターデザインをしているのに，服にオリジナリティがないと片手落ちのような気がするのです。シャネルとかクリスチャンディオールなどのデザイナーズブランドを着ているといった設定時はきちんとデザインしますが，他人がデザインしたのと同じ服をそのまま着せてあげても自分の個性が出ませんし，デザインしている自分だってつまらないからです。それに，ぴったりキャラクターの雰囲気にある服のサンプルなんてのは普通なかなか見つかるもんではありません。

ですから，資料はあくまで参考レベルにし，自分なりのアレンジを加え，オリジナルをデザインするようにつとめます。オリジナルとは服そのもののデザイン全体を考えるのもいいですし，形は既存のものを使い，模様部分のみをオリジナルのものにするとか，既存服についていた小さなアクセサリを思いっきり大きくして存在感を持たせるとか，襟や袖の部分だけを変形させるなどです。とにかく，一部でも自分がデザインしたオリジナル部分があればいいと思っています。完璧とか実用的とかなんて言葉には半分目をつむって自分の好みなデザインをしています。「私はこういうデザインの服がほしいんだぁ！ 作りたいんだぁ！」とこだわってデザインするのです。本職デザイナーから見れば笑っちゃうようなデザインなんでしょうが，自分なりのアレンジってのがあって初めて「このキャラクターはあたしがデザインしてあげたんだ～」って実感がわくのです。そうすると自分もうれしいですし，キャラクターも喜びます。「自分らしさ」というのは服をデザインするときも大事な要素なのではないでしょうか。

服のデザインってのは確かに大変な作業ですが，「キャラクターの系統を知る」「既存製品にとらわれすぎない」「自分のこだわり（オリジナリティ）をデザインする」てのをやってみると気楽にデザインできるようになります。私も自分のデザインが完璧とは思っちゃいませんが，あれこれ悩みながらも楽しんでやってます。これからも，自分もおしゃれをして，原宿とか表参道あたりに行っていっぱい遊んで，ついでに勉強して，どんどんオリジナルの服をデザインしていきたいもんです。では。

●参考雑誌
「ViVi」（講談社）
「ファッション通信」（流行通信社）
「ノンノ」（集英社）
「COSMOPOLITAN」（集英社）
「FIGARO」（TBSブリタニカ）
「Hanako」（マガジンハウス）
「CLASSY」（光文社）

■聞かせてください

女の子のファッションについての悩み，質問，知りたいことなどがありましたら，メール（Oh!X宛）や読者様はがきでリクエストください。服に関する女性の悩みはいっぱい聞いてますが，男性が持つ女性の服装に対する悩みや疑問ってのは皆無なのでちょっといろいろ聞きたい気分です。さまざまな意見が聞けるとうれしいと思います。

Oh!X-LIVE in '99

要Z-MUSIC ver2.0以降 SC-88対応
FINAL CHANCE
データ制作　松尾直樹

常連さん再び

旧Oh!X時代のOh!X LIVEの常連投稿者である松尾直樹君(オンラインネームはマッチュン)がまたまた久々の投稿をしてきてくれました。

旧Oh!Xが休刊後，マッチュン君はZ-MUSICを使って音楽制作を楽しむユーザーのためのホームページ「Z-MUSIC推進委員会」(http://www.asahi-net.or.jp/~PT6N-MTO/)を開設し，Z-MUSICユーザーのためにさまざまな有益な情報を発信してくれています。「Z-MUSIC推進委員会」のBBSにはマッチュン君以外の著名な音楽データ制作者たちやユーティリティプログラマたちも訪れてきますのでZ-MUSICユーザーの情報交換の場としてはもってこいです。Z-MUSIC対応の音楽データなどもアップロードされていますのでZ-MUSICユーザーは要チェックのサイトですよ。なお，このほかZ-MUSICの話題を中心にしたサイトとしては，

「mu-Z studio」(http://www02.so-net.ne.jp/~mute/)…mute氏制作

「砂の城」(http://www.imasy.or.jp/~jimbe/)…じんべ氏制作

「わいやぎのwebページ」(http://www2s.biglobe.ne.jp/~yyagi/)…わいやぎ氏制作

があります。かくいう私も「Z-MUSIC HOME PAGE」(http://www.z-z-z.gr.jp/zmusic/)なるページを作っていますので見にきてやってください。

カシオペア，アルバム「FULL COLORS」より「FINAL CHANCE」をお届けします

話がそれました。採用曲の解説に参りましょう。

今回採用となったマッチュン君の作品は，日本を代表するフュージョンバンド「カシオペア」の「FINAL CHANCE」という曲のコピーです。この曲は1991年に発売されたアルバム「FULL COLORS」(PICL-1016/パイオニアLDC)の5トラック目に収録されています。このアルバムはカシオペアファンはもちろんDTMファンには相当な人気があるらしく，インターネット上やパソコン通信上にはこのアルバムの収録曲のコピー曲が多数アップロードされています。

なかでも人気が高いのはテレビ番組や各種イベントなどのテーマ曲として使われたことのある「FIGHTMAN」「NAVIGATOR」の2曲です。実際「FIGHTMAN」はZ-MUSIC ver.1.0対応でローランドSC-55用データとして旧Oh!X 1993年4月号のOh!X LIVEに掲載されたこともあります(データ制作＝乾圭典)。

さて，今回マッチュン君が制作してくれた「FINAL CHANCE」は人気曲ではないですが，なかなかの名曲です。この曲に目を付けるあたり実に渋いですね。この曲はカシオペアのキーボーディスト向谷実作曲なんです。

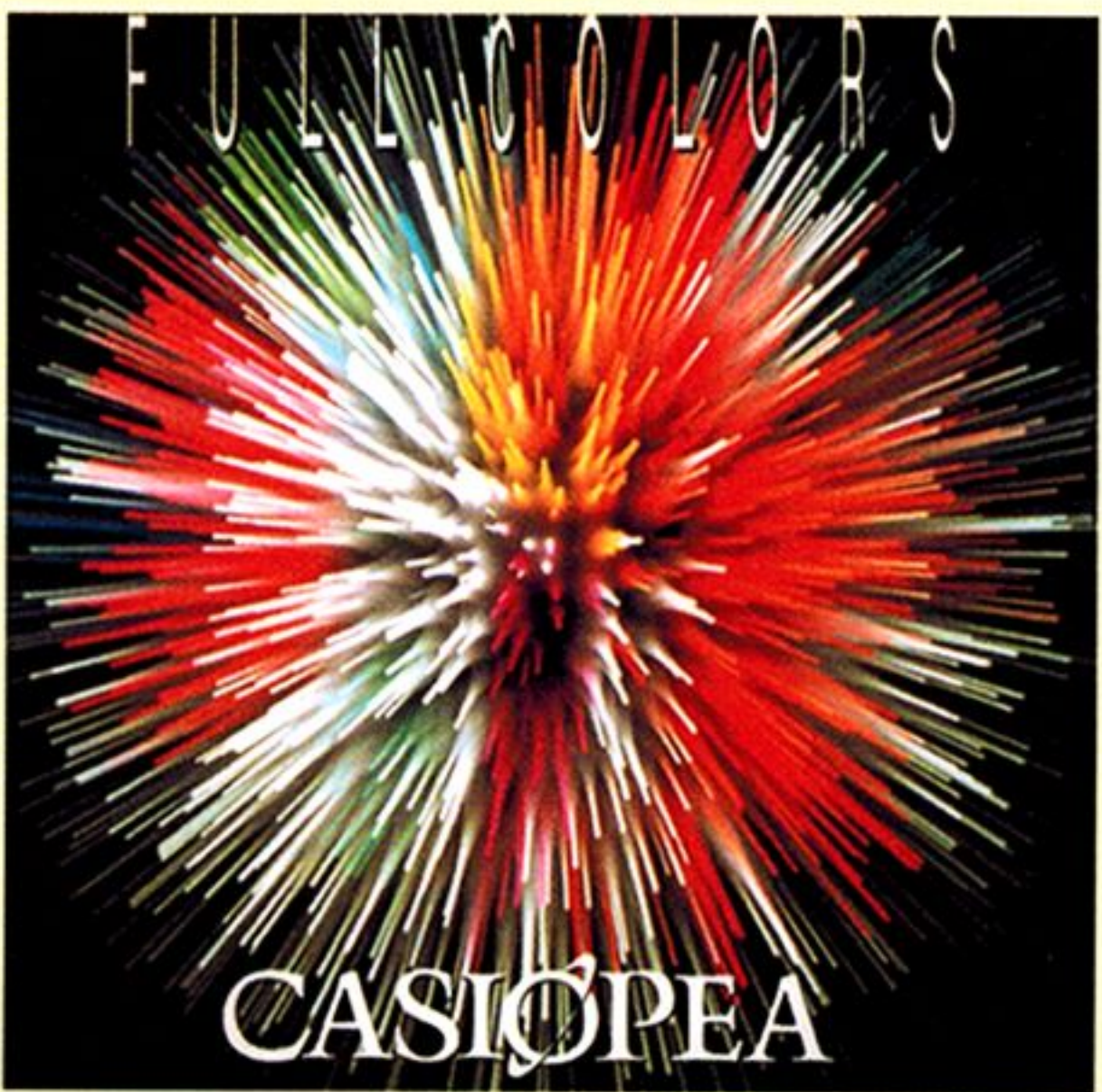
カシオペアの曲というとキャッチーなメロディが心地よい，リーダーを務めるギター担当の野呂一生作曲の曲ばかりが取り沙汰されますが(先に挙げた2曲の人気曲はいずれも作曲は野呂一生)，シンセサイザ(キーボード)フリークやDTMマニアは向谷実の曲に惹かれるんです。

というのもあの技巧的なコードワークとアグレッシブな曲調は，曲データ制作を趣味とする我々には非常に挑戦しがいがあるからなんですな。

特にコードワークの部分は，コピーをしていると非常に勉強になります。「こういう和音をこういうときに使うのか」というのが打ち込んでいるときに次から次へと発見できるんですから。

また，メロディ主体の曲ではどうしてもバックバンドの部分の打ち込みがいい加減になりがちですが，向谷音楽ではそうはいってられません。各パートの楽器が複雑に絡みあって作り出す一種独特のハーモニーの中で旋律(とおぼしきもの?)が作り出されていくので，各パートの各音のベロシティのばらつきや，ゲートタイムの切り方がちょっと違うだけで原曲とは感じが違ってきてしまうんです。ですから市販されているバンドスコアを見て打ち込むだけでは再現できないわけで，データ制作者側にはかなりの音楽分析能力が要求されてきてしまうのですね。まさに上級DTMファンだけがコピーの挑戦を許される，実に挑戦し甲斐のあるチョモランマ級ミュージックという感じです。

今回掲載されているマッチュン君のデータは非常に完成度が高く，市販のDTM曲データ集のクオリティを凌駕しています。原曲の，ピアノとエレクトリックベースの微妙な主張の緩急もZ-MUSICのベロシティシーケンス機能を効果的に使っており，原曲の雰囲気はほぼ完璧に再現されているといっていいでしょう。

SC-88VLで演奏させたところ，音色の若干のニュアンスの違いからかバックでリズムを刻むナイロンギターの音量がちょっと大きい気がしました。こちらは入力するときに各自調整するといいかもしれません。

トラック1の先頭に，

[track_fader 0,80]

を挿入するとより原曲に近くなる気がします。

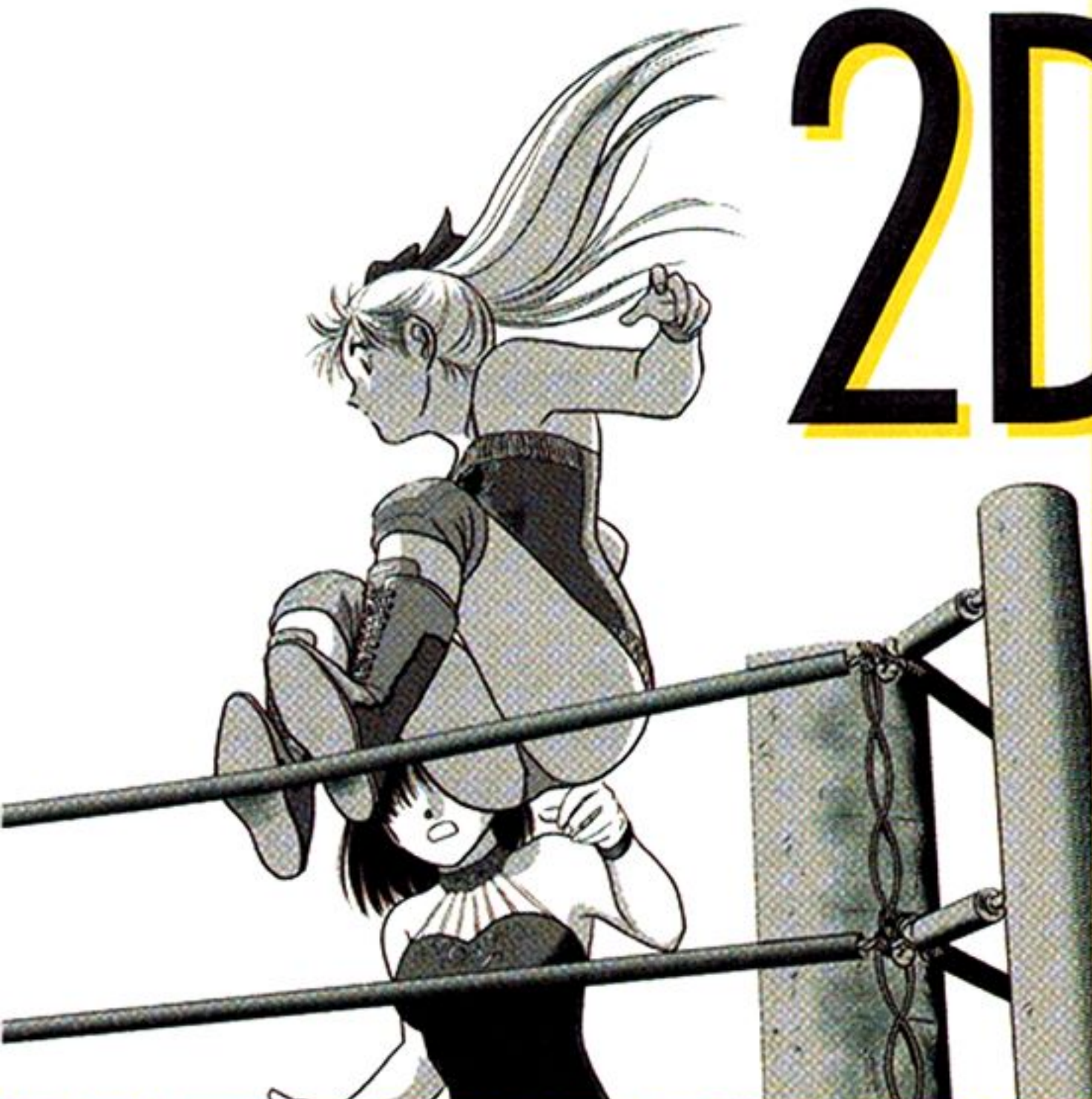
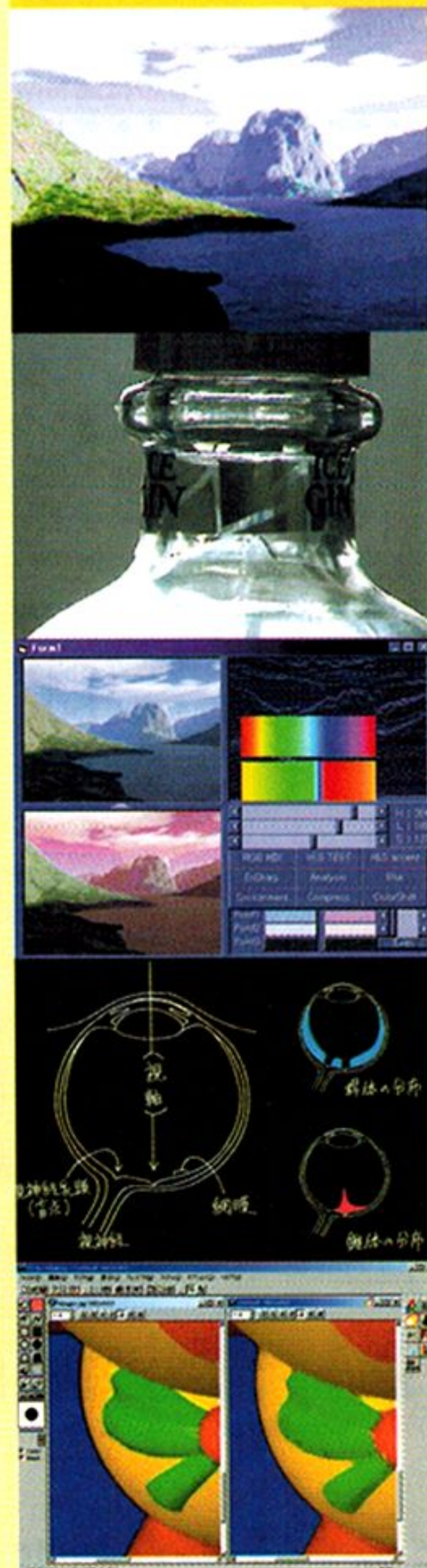
聞きどころはやはりトラック3のピアノ，トラック6,7のベースでしょう。ZSVなどを使ってこの3つのトラックのみを演奏させると，マッチュン君の打ち込みテクのすごさと，向谷実の和音テクの巧みさの両方が一度に学べるはずです。

曲データはSC-88/VL/PRO，SC-8850に対応。演奏にはZ-MUSIC ver.2.0以上が必要です。もちろんver.3.0でも正しく演奏されます。

(西川善司)

Contents

特集

2D Laboratory

2D系のグラフィックというのは，Oh!Xのテーマであるプログラムなどのロジカルな部門でも，またアートな部門でもかなり大きな意味を持っている。グラフィックツールやフィルタに関するさまざまな試論は旧Oh!Xでもしばしば展開され，実際に試す場所としてEX Systemが作られた。アンチエイリアス，マッピング，モーションブラー，印刷時の誤差拡散処理やカラーマッチングなど，現時点で見ればありがちなものも多いが，それなりに先見性を持った展開をしてきたと思う。
当時のX68000という環境がかなり制限を伴ったものであったのに対し，MacintoshやWindowsなどの環境ではグラフィック処理上の制限がきわめて緩くなってきている。新しい次元でグラフィック処理を展開していくための第一歩を踏み出していこう。
3Dグラフィックとの融合や4Dグラフィックへの展開なども，2D系のベースがあってこそ実現されるものだ。2D処理は思想の問題，フィルタ処理はアイデアの問題に帰着する。今後の展開の方向を見極める意味でも現状を見直し，さまざまなものの可能性を問い直してみたい。

2Dグラフィックツールの可能性を探る

中野修一 Nakano Shuichi

もっと手軽に使えて，なんでもできて，おまけにフリーというようなグラフィックツールはないだろうか？　フリーのツールはもちろんたくさん存在するのだが，グラフィックツールにはまだ完成された形態がないようにも思われる。次世代のツールのあり方について考えてみたい。

2次元系のグラフィックツールを使っているだろうか？　CGをやる人には愚問だが，一般の人にはむしろ縁遠くなってしまっているのではないだろうか。Webページを作っている人ならある程度は使うだろうし，あとはデジカメ画像のレタッチくらいだろうか。

グラフィック表示はフルカラーが当たり前になって，実によい世の中になったものだが，そういったハードウェア環境が揃っても別段世の中がマルチメディア方面に流れていくわけでもなく，市販されているグラフィックツールに搭載された「かなり凄い機能」などはほとんど使われていない。

にもかかわらず，「PhotoDeluxe使ってるんですがPhotoshopでないとダメですかね」とか「PhotoshopLEでは安心できない」という，よくわからない人も多い。個人がちょっと本格的にグラフィックをやりたいと思うレベルと，世間での本格的なグラフィックのレベルの差が恐ろしく開いてしまっているのに簡易版ツールは敬遠されがちな傾向にある。もちろん，「簡易版」とはいっても普通の人が使うには十分な機能は備わっているし，フルバージョンのPhotoshopを導入したユーザーは機能をもてあます。

もっと一般ユーザーとグラフィックツールは親しい関係にあってもよいはずだと思う。ツールを立ち上げた途端，なにをしていいのかわからなくなるようなツールが多すぎはしないか？　使っていて感動のないツールが多くはないか？　CGで絵を描く人はずいぶん増えたが，都築氏や森川氏のようなパソコンCGの可能性を引き出すような使い方をしている人というのはさほど多くない。

個人的な印象で申しわけないが，グラフィックツールというのは，パソコンを使っていて最初にその可能性を教えてくれるツールのひとつだと思っていた。かつてはMacPaintやDeluxePaintのように，特に絵描きさんでなくても触りたくなるような印象的なツールが多かったように思う（とはいえ私はMacPaintでなにをしていいのかわからなくなった口だが）。

X68000ユーザーなら，Z'sSTAFF PRO-68KやMatierを立ち上げたときのワンダーな展開の予感を覚えてはいないだろうか？

既存のツールについて

グラフィックツールというのはベース部分ができていれば，あとは個々のモジュールの追加で形ができてくるので，規模の割には比較的作りやすいツールだろう。しかし，要求されている項目数がかなり多いのも事実でそれらを揃えていくだけですでにげんなりしてしまいそうだ。たとえば，TWAIN機器への対応が面倒そう。タブレットとかへの対応が面倒そう。あまり使いではないけど，一応揃えておかなければならないフィルタ群……。

逆にいえばひととおりのものを揃える根性があれば，成り立ってしまうものでもあるのだ。

Windows95登場当時はグラフィックツール自体が珍しかったためかDaisy Artが注目されていたが，ペンの動作などを見ると，とうてい納得いくようなものではなかった。マウスをちょっと押すとどんどん上書きされて濃くなるので，最初は濃度指定がきかないのかと思っていたくらいだ。マウス位置を動かさなくても描画されるというのは相当慣れを必要とするのではないかと推測される。高速マシンで使用してはいけないのはいうまでもない。

この辺がWindowsのグラフィック環境に関して疑念が浮かんできた最初だったかもしれない。どうも理屈で作られたツールのにおいがする。Oh!X復刊号にはなんと3種類のグラフィックエディタが収録されていた。ユーザーメイドのツールにはそれなりに主張があって，機能的にも面白いものがある。いくつかのツールについて見てみよう。

●LightEDitorの構造

いきなりだが，わかりにくさは天下一品だろう。最初にマニュアルのBMPファイルを読むのが大切。単純に筆とかを選んでいるだけならまだいいのだが，機能の組み合わせになると，なにが起きるのか予想がつかない。機能の直行性もいまひとつなので基本的には覚えるしかない。組み合わせによる機能の自由度は非常に高いのだが，使いきれない……。ウルトラ簡易バージョンがほしいところ。

特筆すべき点は多いが，ここではペンによるエフェクトを挙げておきたい。エフェクトを伴うもの自体もペンと定義されているのだ。また，あちこちのパネルによる指定はさまざまに絡み合って複雑な効果を出してくる。

この方向性は，機能や色などの指定までの流れを考えると，複雑な動作にもかかわらず，基本構造自体はむしろシンプルにできるように思われる。関係ないけど，シンセサイザを使ったことがある人には自然な構造に映るかも。

●EX for Winの操作性

開発段階から口が出せるので，私にとっては理解しやすい。慣れているだけという話もあるが……。気に入らない点も多々あるのだが，ほかに改良すべき点が山ほどあるためユーザーインタフェイスなどを改善していく余裕がないのが残念だ。以前のX68000版EX Systemではユーザーインタフェイスについては最初から捨てるというスタンスで機能開発中心に展開していたのだが，Windows版ではもう少し考えたかったところ。ただし，ペンを中心にした操作性はかなり完成されたところにあると思う。右スポイト，シフトUNDOの使い勝手はレタッチ作業などにもかなり有効だ（UNDOの概念にやや問題があるのだが）。

パターンブラシと通常のペンは一括管理されるべきものだが，濃度バッファの有無など，内部処理的にはかなり違うので現状の構造からは同一にするのが難しいという。うーむ。

図1　Light EDitor

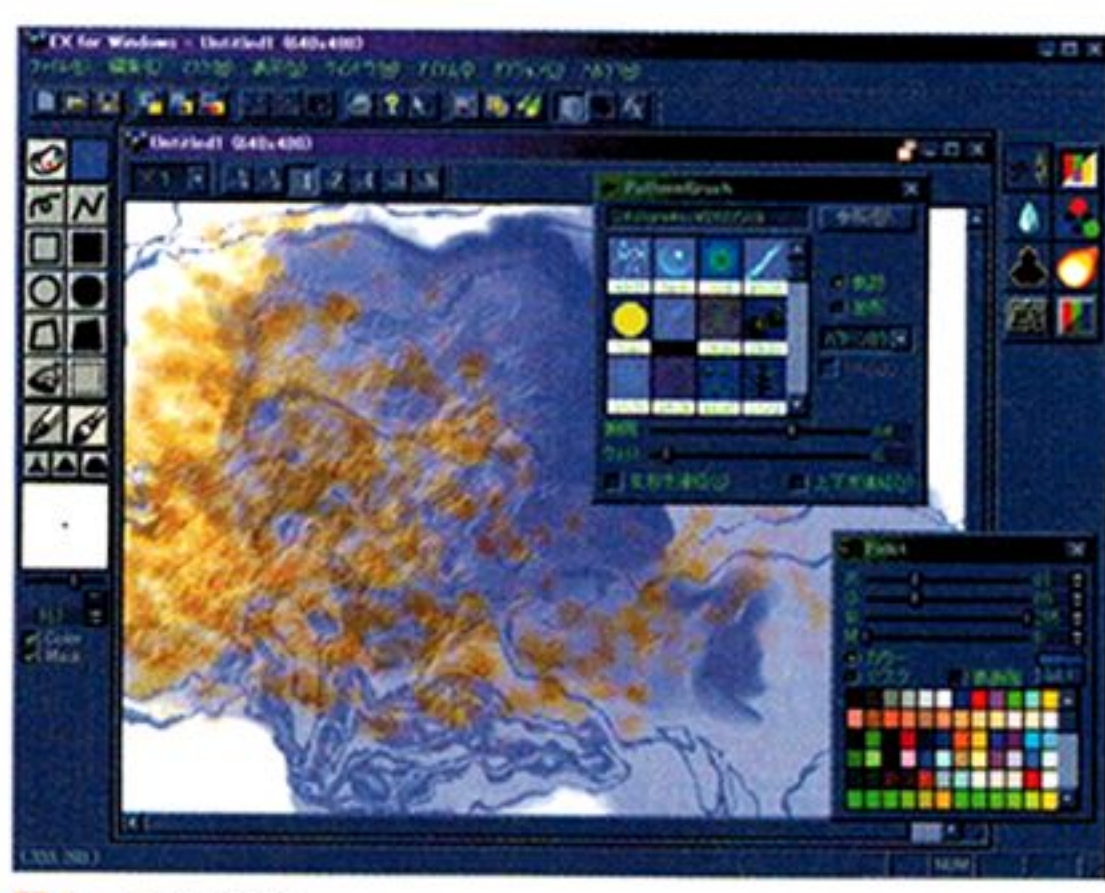



図2　EX for Win

現状での問題点は機能が少なすぎるということ。「ンなものいつでも作れる」系のフィルタがないため，ちょっとしたことでもほかのツールが必要になる。あとは対応ファイル形式の問題か。

本体機能に匹敵するものをプラグインで構築できるのだが，機能の呼び出し方などは差別化されており，いまひとつ綺麗ではない。足りない機能がプラグインで揃うと，それはそれで破綻しそう。基本システム＋総プラグイン化で再設計というのが手だろうか。

なお，Photoshop互換のプラグインというのは業界では標準となっているのだが，EXでは対応していない。それを開発しようとするとPhotoshopが必要になるというのは困った問題だ。本来は当然取り上げるべきテーマのひとつなのだが，SDKがもっと流通してないと意味がないということで見送っている。

●GIMPという選択

GIMPとはGNUプロジェクトの一環で作成されつつあるグラフィックツールで，一部ではPhotoshopに匹敵するといわれているツールである（UNIX環境でだけだろうが）。

X Window特有の使いにくいユーザーインタフェイスなど，よくないグラフィックツールの特徴をかなり備えている。

そもそもオーバーラッピングマルチウィンドウシステムがグラフィックツールにはあまり向いていない（というか，向いている分野のほうが少ないか）というのはX68000でZ'sSTAFFを使っていた人にはよくわかるのではないだろうか。GIMPではそれに輪をかけて，描画中のウィンドウさえ前面に出てくるので，描画領域にはツール類は並べられない。手動でタイリングウィンドウ管理を行うのがもっとも効率よく操作できる環境となりそうだ。なにか本末転倒である。ちなみに，描画ウィンドウが最上面でなくてもクリックと同時に描画は開始される（Windowsアプリでもままあることではあるが）。SX-Windowなどの操作性に慣れているととまどうことがある。

率直にいうと，フィルタの開発環境としてはいいかもしれないが，このシステムは根本から作り直さないとダメだと思う。しかし，GIMPのプラグインを流用するなりコンバートするというのは考慮してもよいかもしれない。

ついでに，現バージョンではよく落ちる。まだまだこれからだ。

フィルタリングの最適化

ツールの内部構造について考えてみよう。

一時，RGBを16ビットずつで処理するべきだという意見を聞いたことがあった。ずっと5ビットでやっていた身としてはそんなに必要なはずはないと思うのだが，画像処理でコントラストなどを大きく変えたあとでほかの処理を行うと階調性が損なわれてしまっているとかいう意見だった。

主に写真の加工でアンダーやオーバーの素材を適正露出にした時点でフルスペックの24ビットデータがほしい，ということだったと思う。ただ，そのためだけに全処理を48ビットで行うのはいささか牛刀だし，元データが十分にハイビットであるのが前提条件だ。それなら写真なら取り込みツール，デジカメならデジカメ側で対応してもらって前処理ツールで加工すれば済む話だ。すべての工程を48ビット化する必要は感じられない。どうせ加工に伴う情報量の低下は避けられない。

フィルタリングを掛け算的に行うというか，一度処理した結果に対して行うのではなく，フィルタ部分の処理をまとめて行う方法もある。グラフィックツールによっては，元画像はずっと保管しておいて，フィルタ処理をレイヤーで扱うものがある。

取り込んだ写真に対し，明るさを半分にしてからコントラスト加工し，明るさを倍にしたとすると，Photoshopなどでは，当然，元に戻したときに階調性が大きく損なわれていることになるが，フィルタ処理はフィルタ処理としてまとめておいた場合には途中の演算を十分に高精度で行うことで最小限の劣化だけで画像を加工できる。確かに計算コストが見合えば悪い方法ではない。

もちろん，素直にやると重い処理を施された画像をいじることは困難になるので，レイヤー構成はレイヤー構成として記録しておいて，描画はビットマップだけで処理しておいて，最終的にレンダリングし直すなどの措置は必要になるだろう。「画像の保存は手順ファイルで行ったほうがいいのではないか？」ということにもなるな。

フィルタ適用量と機能をそれぞれ分離してレイヤー化するなどといった構成はきわめてコンピュータ的で美しい。アルゴリズムや制御構造などを取り入れるとさらに面白いことになる。単純なフィルタでもフィードバックループを組むだけで非常に多彩な結果を出力してくれるだろう（うまく制御できるかどうかは別問題として）。フィルタ自体がプログラマブルであるという自由度は重要だ。

データ量の適正値は？

RGB48ビットは置いておいて，画像データはRGB24ビットでよいのか？　という疑問もある。RGBα32ビットという選択はもはや常識だろう。

透明度があるなら反射率がないのはおかしいとか，屈折率も必要だとか異方性反射だったらどうするとか，周波数特性は……云々という付随的なパラメータはいくらでも出てくる。

そういうのだけでいいのか？　まず，Zがほしいという意見もある。2DツールがZバッファを持つとどうなるだろう。ZつきのパターンをZつきのバッファに合成することを考えよう。空間の奥行きに従って立体的な配置ができる。2Dのクリップアートを使っていくのとはちょっと違う次元のことができる。また，フォグ系の処理を加えるのも簡単だし，被写界深度（今回はほかの記事でたくさん説明があるだろう）の設定も可能になる。

32ビットZ，8ビットステンシルなどというとどんな処理をするんだという話になるが，さらにいえば法線ベクトルがドットごとに保存してあれば，光源処理や素材指定に頼らないキャンバス処理など質感表現でも応用が可能になる。

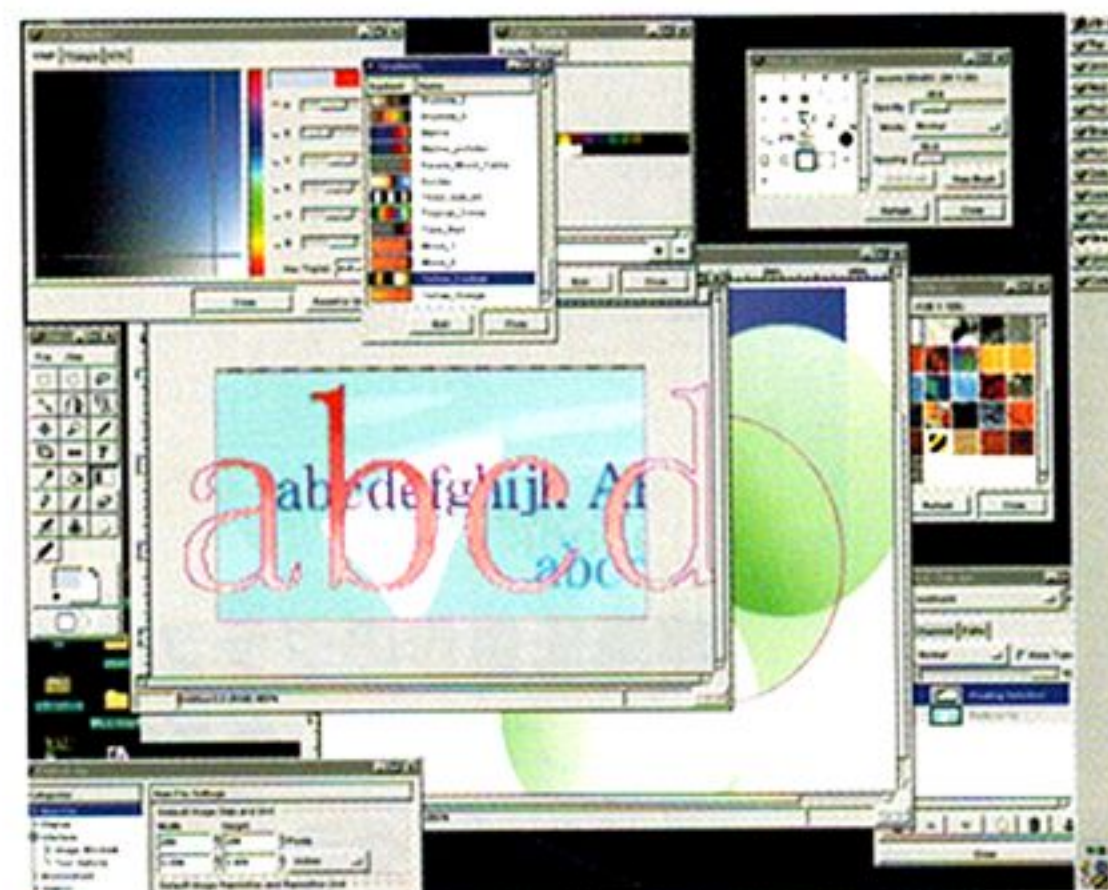



図3 GIMP

また，ドットごとに方向性（筆の方向や物体の移動ベクトル）を記録しておけば油絵変換などでのタッチの再現やモーションブラー処理などへの応用が利く。これは2次元処理なら8ビットの極座標と8ビットくらいの速度，あるいは3次元に拡張して8ないし16ビットずつのX，Y，Z値だ。

このように考えていくと，簡単に1ピクセル256ビットくらいにはなってしまう。しかし，これらのうち常に使用されるものというのはさほど多くない。すべて持っているのがいいのだが，必要なときだけパラメータを拡張するのがいいだろう。そう考えると，先ほどのフィルタの扱いのように，レイヤーとして定義するのがよいのかもしれない。データのフィールドがいくつかあって，どのような解釈によるものかはそれぞれが別途指定するという感じだ。当然，ピクセルの構成はプログラマブルになるということである。

色なり，フィルタへの入力値なりは等しくフィルタによる処理を通過する可能性があるとしよう。これは多彩な効果が得られるものの，わかりにくくなることについてもすでに実証済みなわけで，どれだけ感覚的に使えるものに仕上げるかが重要になってくるだろう。

アニメーションへの対応

アニメーションというのも2D処理が結構必要とされている分野である。ムービーファイルなどを放り込むと，1フレームずつエディットできて，そのままアニメーションさせて確認できるとかいう機能があると，非常に便利だ。

さらに連続したシーンに対してエフェクトを連続的にかけられるとかいったAmigaのDeluxe Paint並みの機能があると2Dツールは新しい次元のツールとなる。AfterEffectsやAuraなどはちょっと高価すぎて手軽に面白いことができるという感じではない。DeluxePaintやEPA2が必要なのだ。

こういったツールも多数の画像を系列的にハンドリングする部分とムービーを編集する部分さえ加えれば2Dツールの守備範囲内といえる。

＊

という感じで，なんとなく次世代EX Systemの構想はふくらみつつあるのだが，かなりの資源を使うとか，開発者多忙とかでなかなか進まない。ま，時期がくるまではじっくり基礎実験を行っておくとしよう。

炎を描く

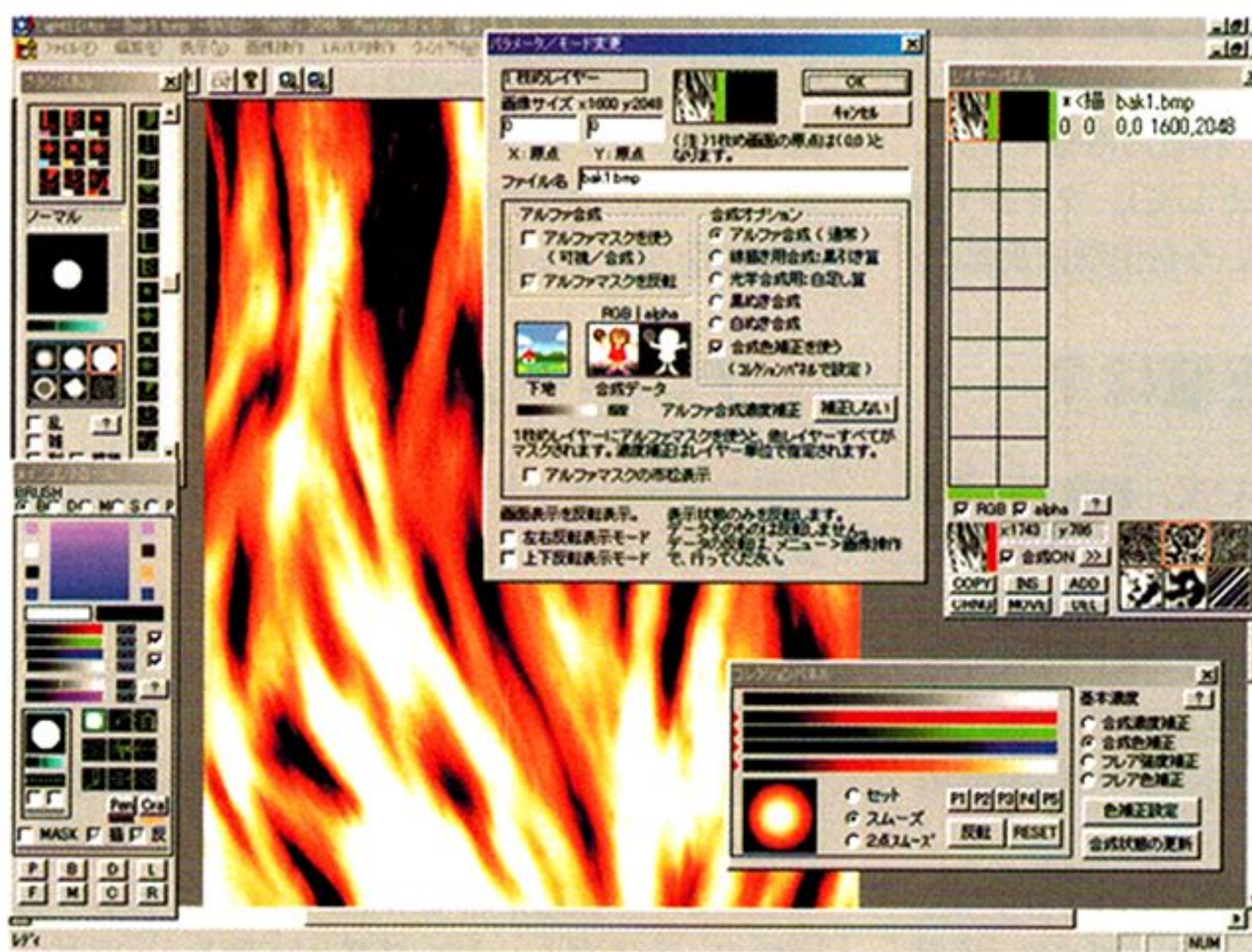
都築和彦 Tsuzuki Kazuhiko

都築氏自作のグラフィックツールは"LightEDitor"という名前のとおり，多彩な光学的表現を得意としている。ここでは，2D表現の難しい「炎」を題材に，エフェクトの使い方や描画の進め方などをプログラムの処理内容にまで踏み込んで解説してみよう。

今回は燃え盛る炎の中にたたずむ少女をイメージしてみました。炎の表現と合成具合の解説をしたいと思います。

私は自作のペイントツールLight EDitor（きらきら筆）を使ってますが，ほかのツールでもほぼ同じことができると思いますので適当に読み替えてください。

まず，炎のイメージをモノクロで描きます。

LightEDitorのコレクションパネルの合成色補正設定で，R，G，B階調の変化度合いを変更します。グレーのグラデーションバーが基本階調（0-255）です。その下に，R，G，Bそれぞれの基本階調と，R，G，Bを合成した階調が表示されています。

初期値では，基本階調と同じグレーの階調になっています。炎らしく，黒＞赤＞橙＞黄＞白という感じで，色の階調をエディットします。

新しく設定されたグラデーション階調を参照して，たとえば，R，G，B各値が灰色（128，128，128）なら，橙（236，121，58）という値に補正されます。

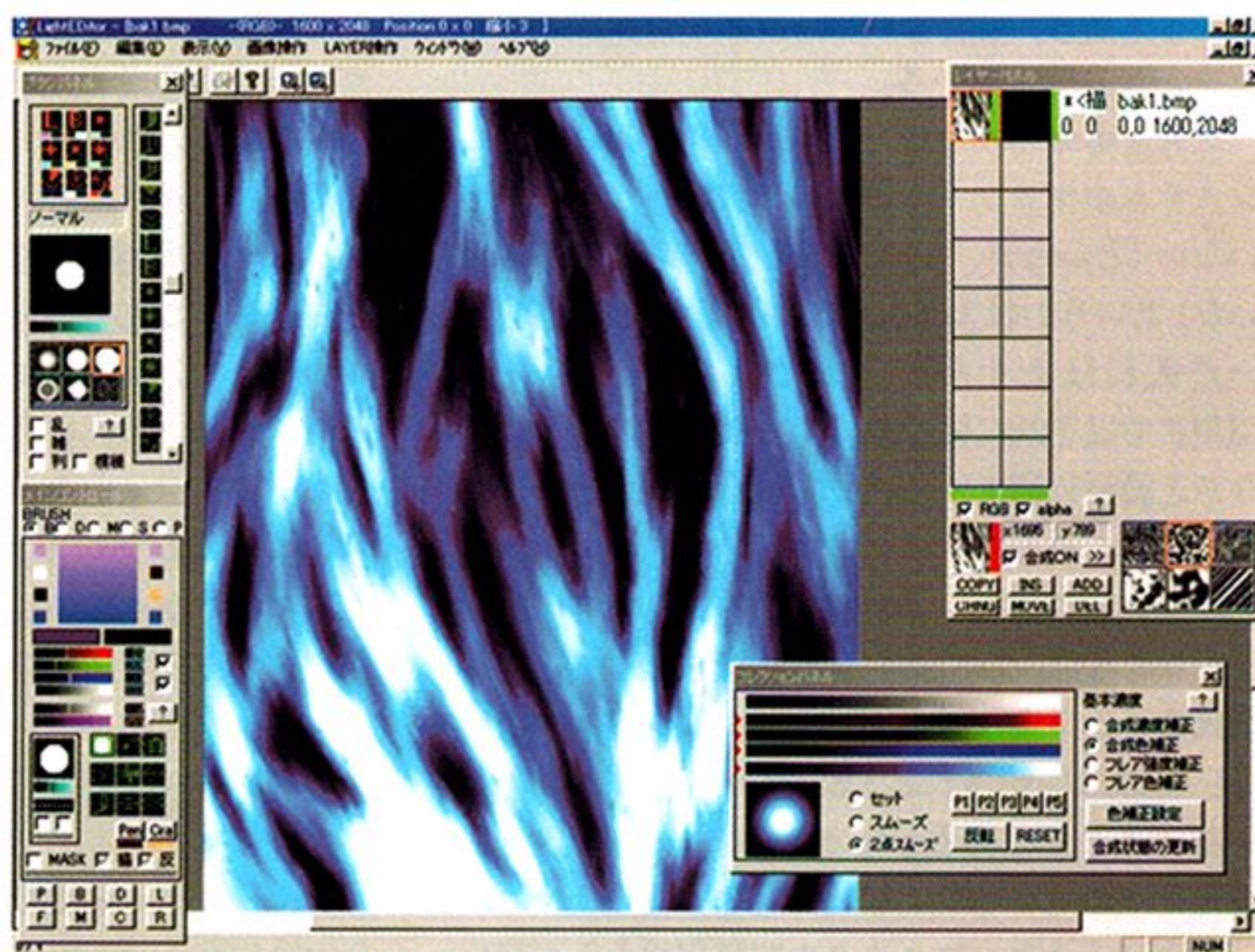
画像に適応させると，モノクロの絵に，炎らしく色味がつきます。具体的には，レイヤーパネルから，ファイル名をマウスクリックして，パラメータ/モード変更パネルを開き，「合成色補正を使う」をチェックします。レイヤーパネルに戻り，「合成ON」させると，合成色補正が適応されます。

この状態で，画像をエディットすることもできます。筆で描画するときは，カレントカラーを炎の色ではなく，グレーで描きます。今の状態は画像そのものはグレーのままで，表示だけ色味がついているからです。

炎などの場合，外側から中心に向かって黒＞赤＞橙＞黄＞白と色が変化します。これを普通に筆で描画しようとすると，カレントカラーの色を変更させながら描くことになりますが，色が変に混じったりとかしてしまいます。

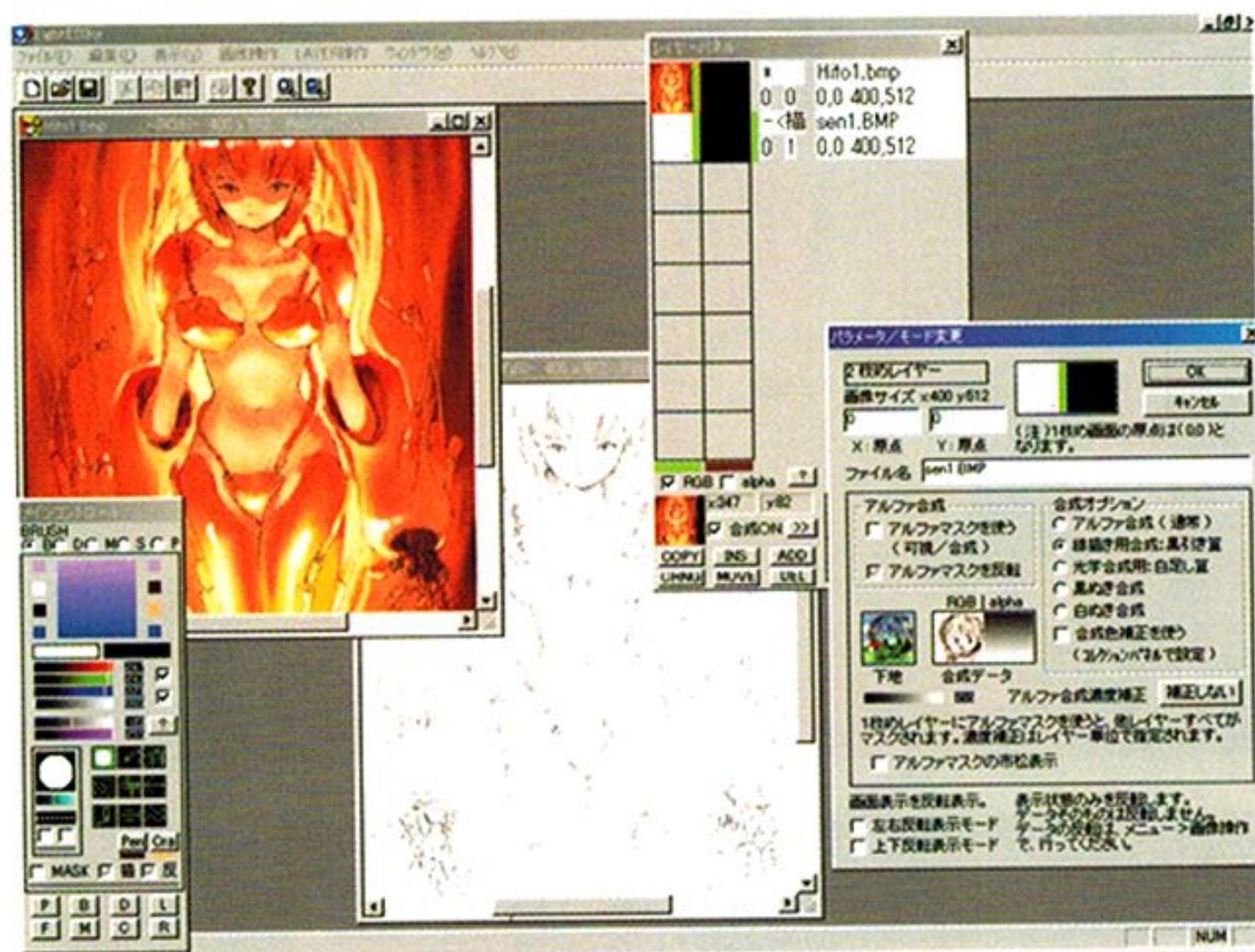
このモードを使えば，グレーの階調を内部で炎の色彩に変換して見せてくれるのです。

たとえば「ひきずり筆」でなじませたときでも，自動的に炎の色味がついて炎が描けます。

また，炎の色味を青っぽくしたいなら，コレクションパネルの色補正をそのように設定するだけで，簡単に変更できます。

炎ができたら，「メニュー＞レイヤー操作＞COMPLETE」で，絵を完成させます。これで，グレーの絵が，炎の色味のついた絵となります。

線画をスキャナで読み込んだものです。

大きいサイズで描くと重いので，sen1.bmpを縮小して人物の下絵を描きます。このとき，レイヤー合成で，「線描き用合成」モードを使います。

下地に線画が合成されます。「線描き用合成」とか，ずいぶん適当な名前をつけてますが，ほかのツールの焼き込み機能と似たような効果です。計算式は違うかもしれませんが。

式で書くとこんな感じです。

basecolor＝下地画面の色
layercolor＝合成画面の色

合成結果＝basecolor－(layercolor×basecolor÷255)

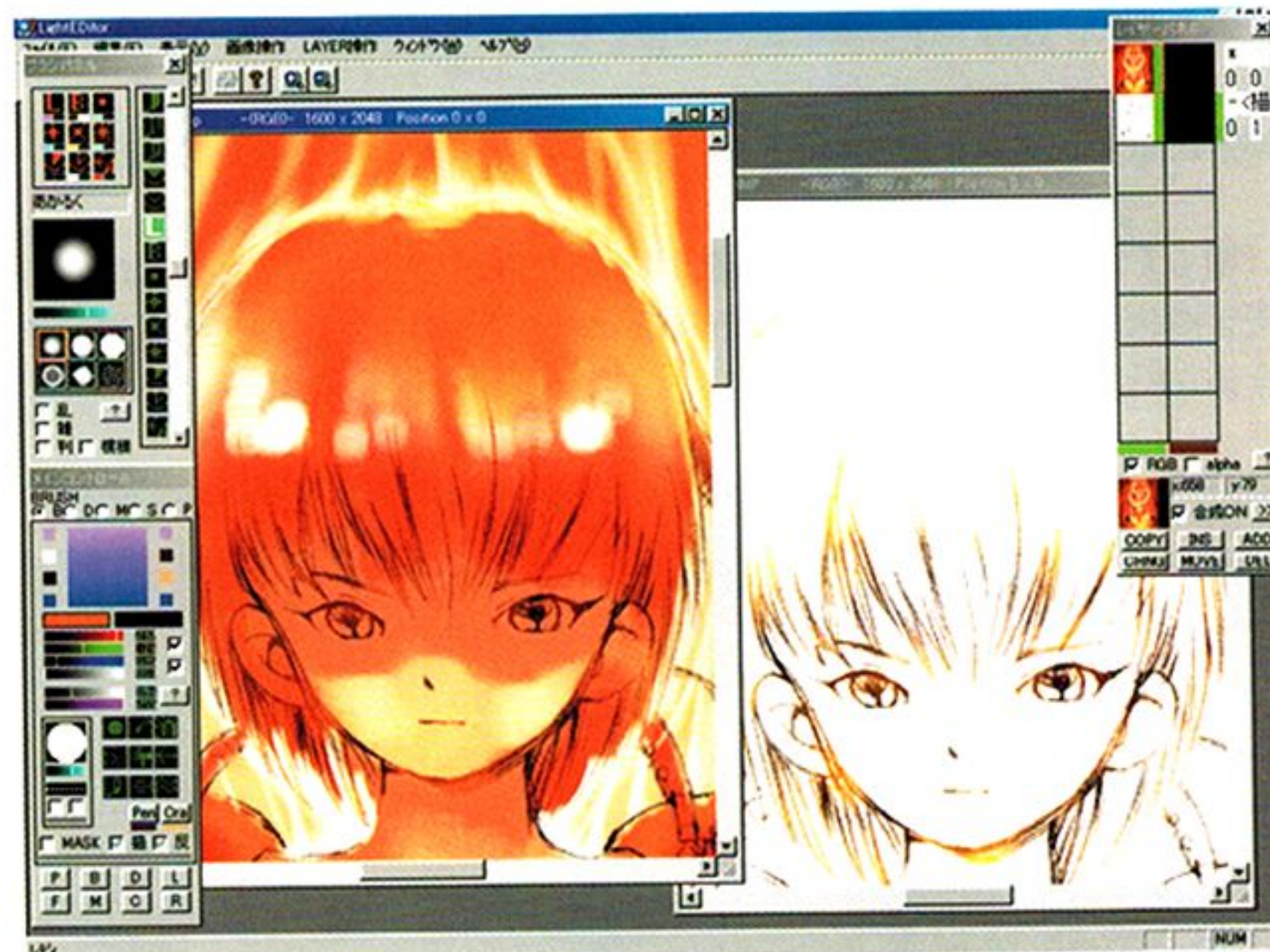
これで，線画の線の色が下地に合成されます。線画のほうが明るく，下地が暗いときでも，より暗く合成されるので線画が見えなくなることはありません。

ちなみに，普通のレイヤー合成は，アルファデータを使います。アルファデータを使った合成については後述しますが，式はこんな感じです。

alpha＝アルファデータ
アルファ合成結果＝basecolor＋((layercolor－basecolor)×alpha÷255)

「線描き用合成」と組み合わせることもできます。アルファデータの値の高い部分だけが合成されるようになります。

線描き用合成結果＝basecolor－((layercolor×basecolor÷255)×alpha÷255)

hito1.bmpを拡大します。

合成させながら，下絵を描き込むと同時に線画を修正して描き込みます。

線画の修正は，線を整えるとともに，微妙に色にアクセントを持たせるようにします。

全体的に茶色っぽく色補正したあと，部分的に明るくさせたりします。カラーインクで描いたような，線の色の濃い部分，明るい部分を作ります。

明るくするためには，「明るく筆」を使い，オレンジ色で描きます。線の暗い部分がオレンジ色に明るくなります。

式で書くとこんな感じです。

basecolor＝下地画面の色
brushcolor＝描画色

描画結果＝basecolor＋(brushcolor×(255－basecolor)÷255)

実際には描画のときの濃度補正等を計算に加えていきます。

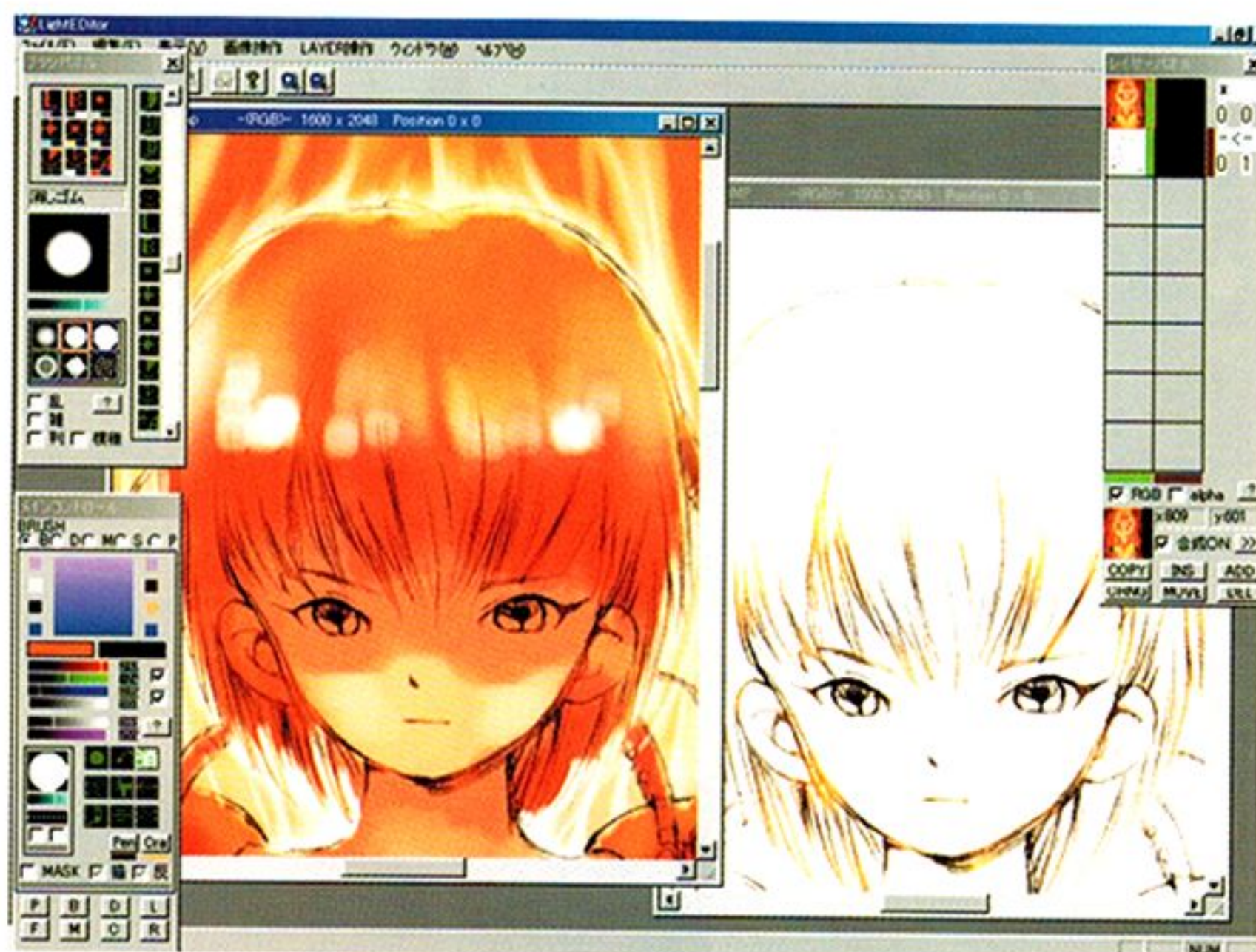
level＝濃度

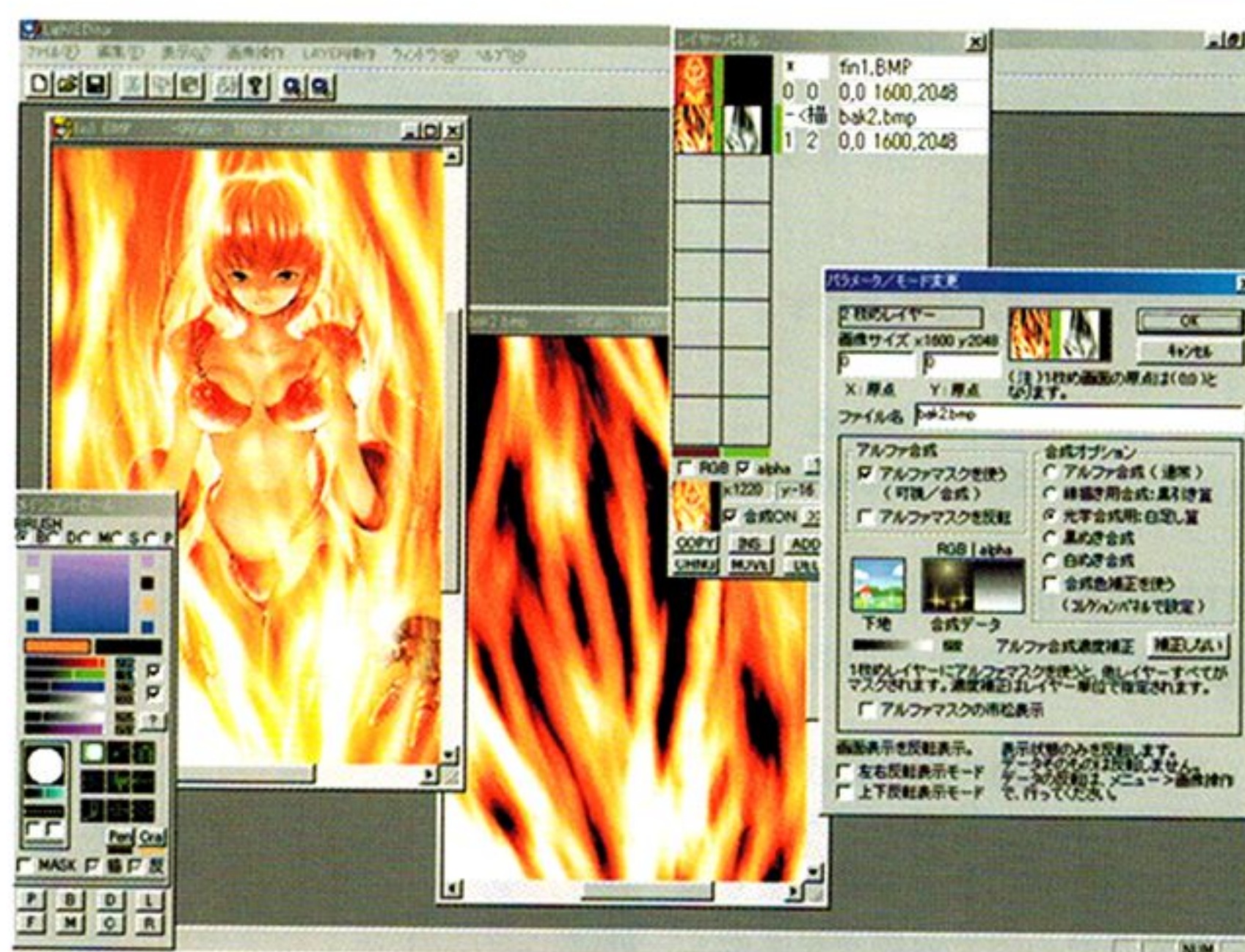
描画結果＝basecolor＋((brushcolor×(255－basecolor)÷255)×level÷255)

これで，線画の線の上を筆で描くと，暗い部分が明るく描画されます。線画より描画色が明るいときでも，明るくなりすぎることはありません。

ちなみに，普通の筆の式はこんな感じです。

描画結果＝basecolor＋((brushcolor－basecolor)×level÷255)

「明るく筆」で描画しすぎて，線画が白っぽくなりすぎたら，「消しゴム筆」で元の状態に戻します。描画前の画像データを保存しておき，そのデータを描き戻しています。

紙の質感を出す「クレヨン筆」で軽く調子をつけてみました。質感パターンはレイヤーパネルから適当に選びます。質感パターンの濃度を計算式に加えると，質感が表現できます。

とりあえずこの辺で合成させた絵を完成させます。完成させてから，さらに細部を修正しています。

fin1.bmpとbak2.bmpを「光学合成」モードでレイヤー合成します。「光学合成」は，「あかるく筆」と同じ式です。他のツールではスクリーンと呼ばれてるようです。ちなみに「きらきら筆」は，「あかるく筆」に十字光パターンを与えたものです。

このとき，「アルファマスク」を使います。合成したとき，人物全体に炎がかかるのを防ぐために，アルファマスクを作り，炎が合成される部分を制御します。

合成させた状態で，炎のアルファ画面を描画します。炎のかかり具合のバランスを見ながら作業できます。

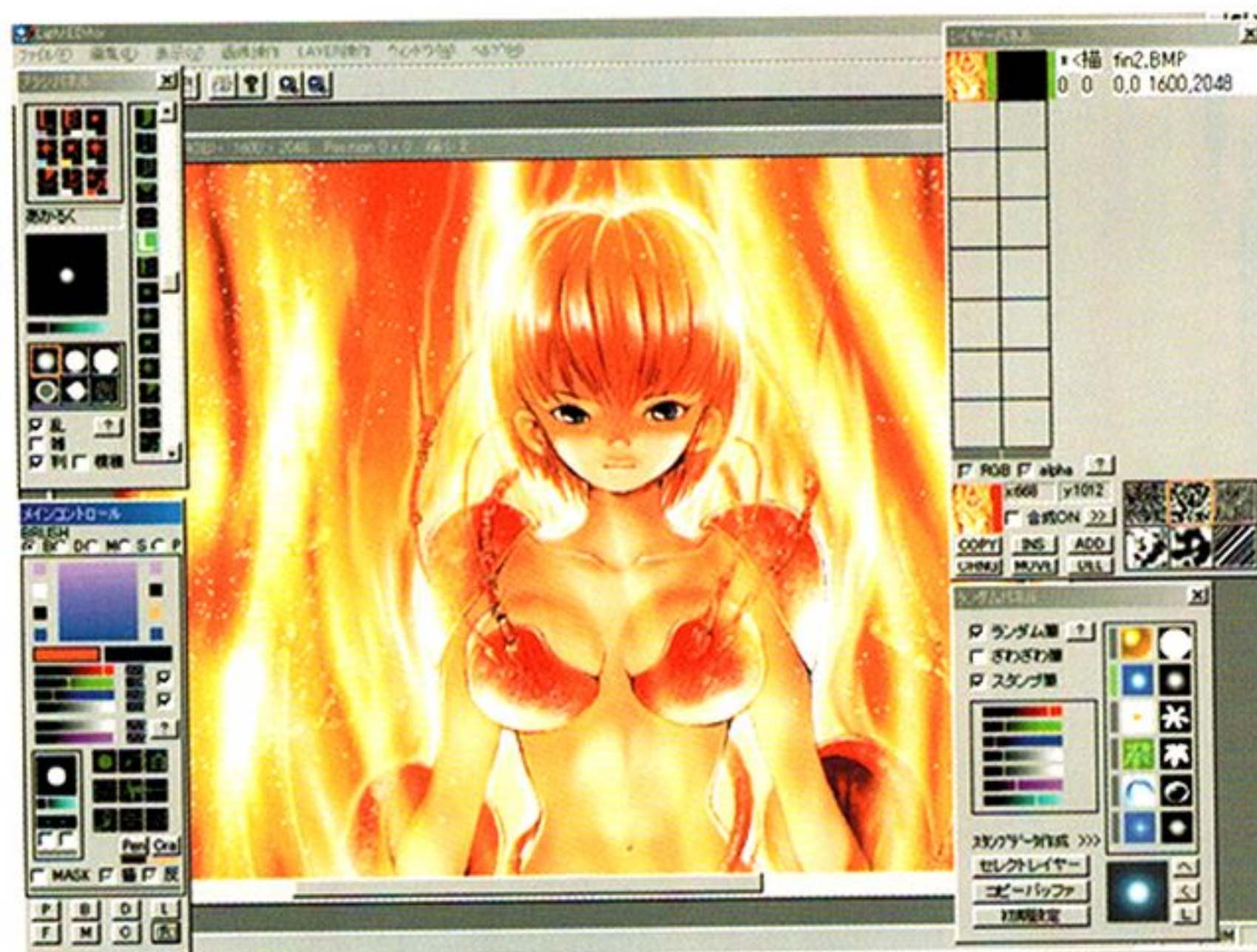
basecolor＝下地画面の色
layercolor＝合成画面の色
alpha＝アルファデータ

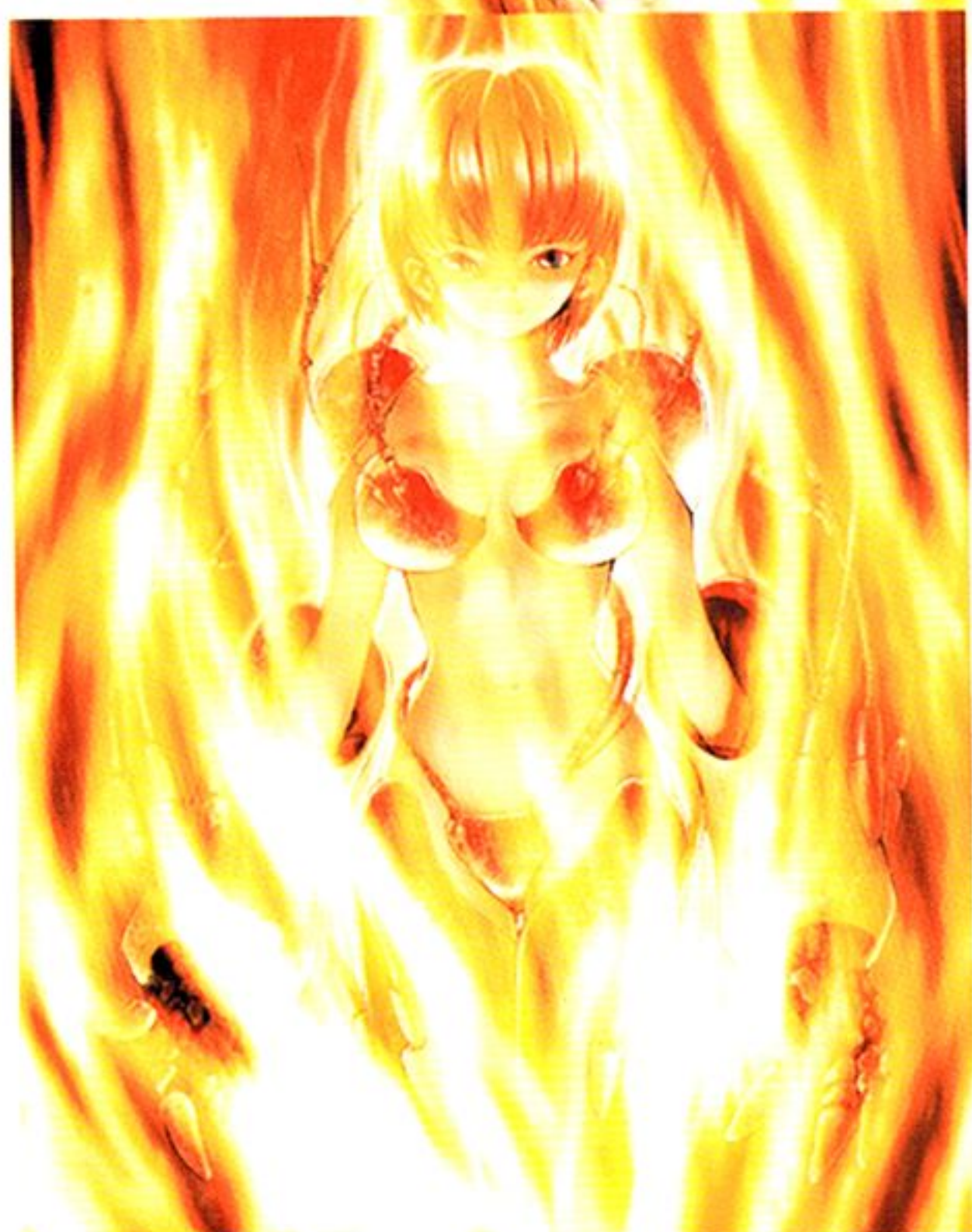
「光学合成」の計算式。

合成結果＝basecolor＋（layercolor×（255－basecolor）÷255）

「光学合成」と「アルファ合成」を組み合わせた計算式。

合成結果＝basecolor＋((layercolor×(255－basecolor)÷255)×alpha÷255)

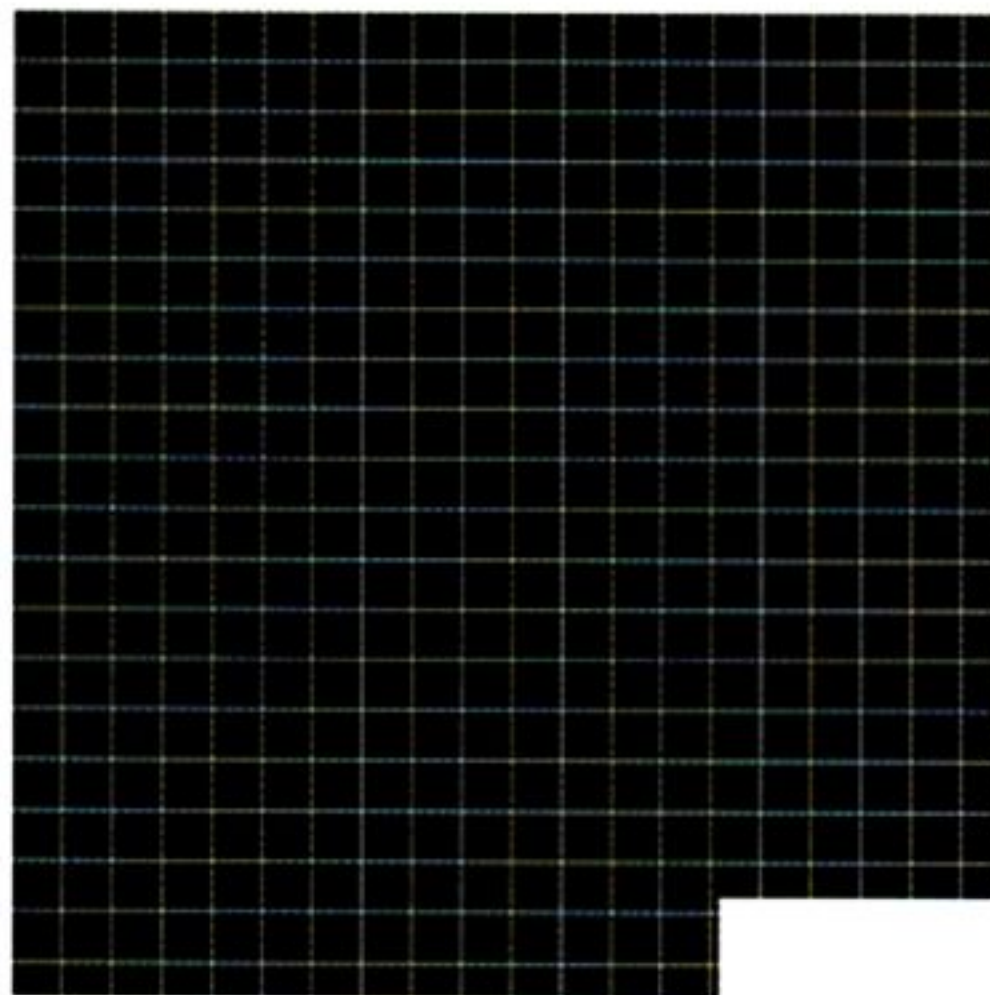
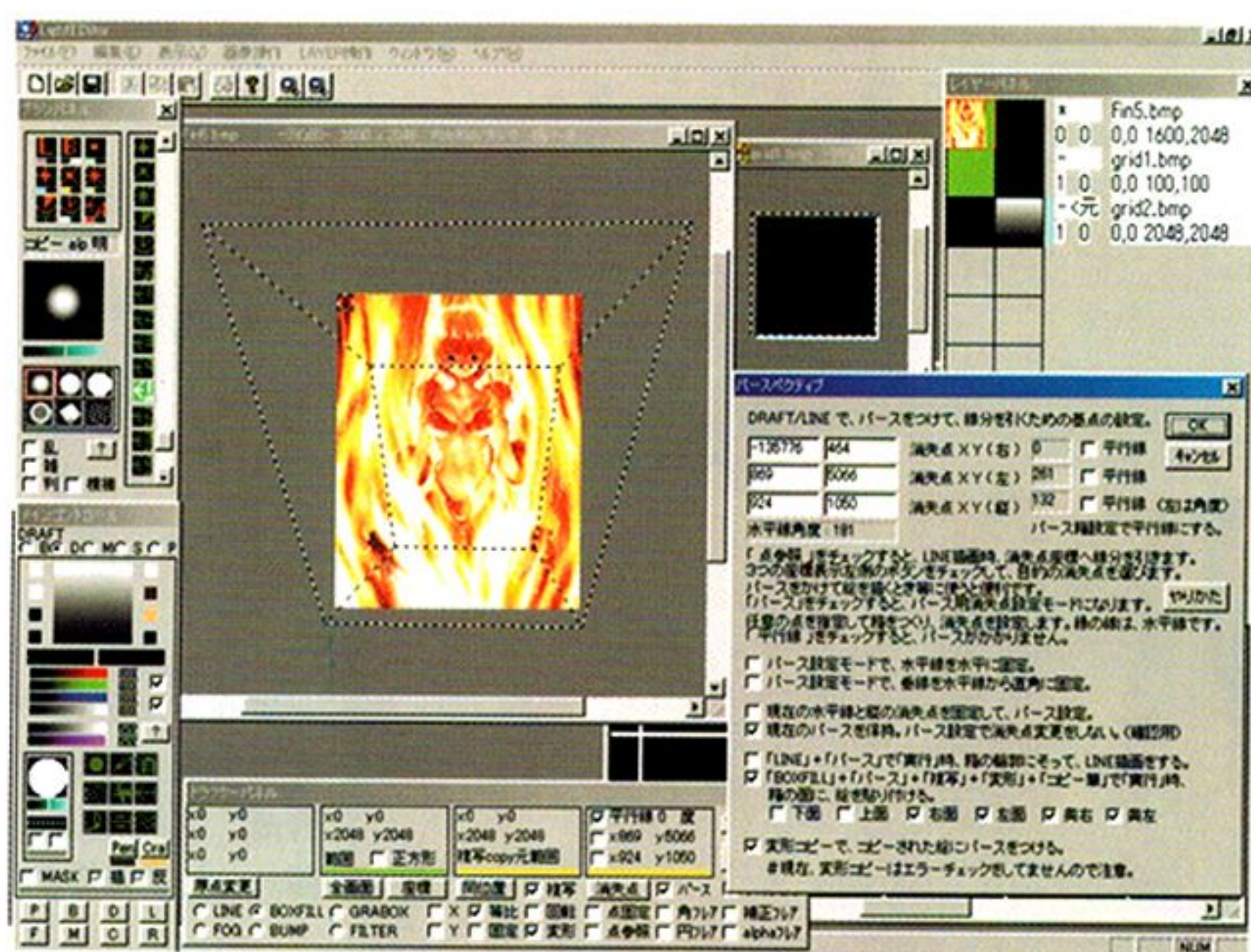
炎のアルファデータを用意します。

アルファデータを使って光学合成。アルファデータの白く見える部分が，合成されます。アルファマスクとも呼ばれます。

アルファデータを使わないで光学合成。人物が炎に隠れてしまいます。

ランダムパネルで，「ランダム筆」「スタンプ筆」を選びます。「ランダム筆」は，筆の描画位置をランダムにばらつかせるものです。「スタンプ筆」は，筆の形状を設定されたパターンで描画します。これで，火の粉をぱらぱらと表現します。

fin2.bmpにフォグをかけてみました。近辺のドットの色の平均をとって，ぼかしています。これと，fin2.bmpを光学合成させます。このときにまたアルファデータを使います。

コントラストは少し高めに補正してみました。

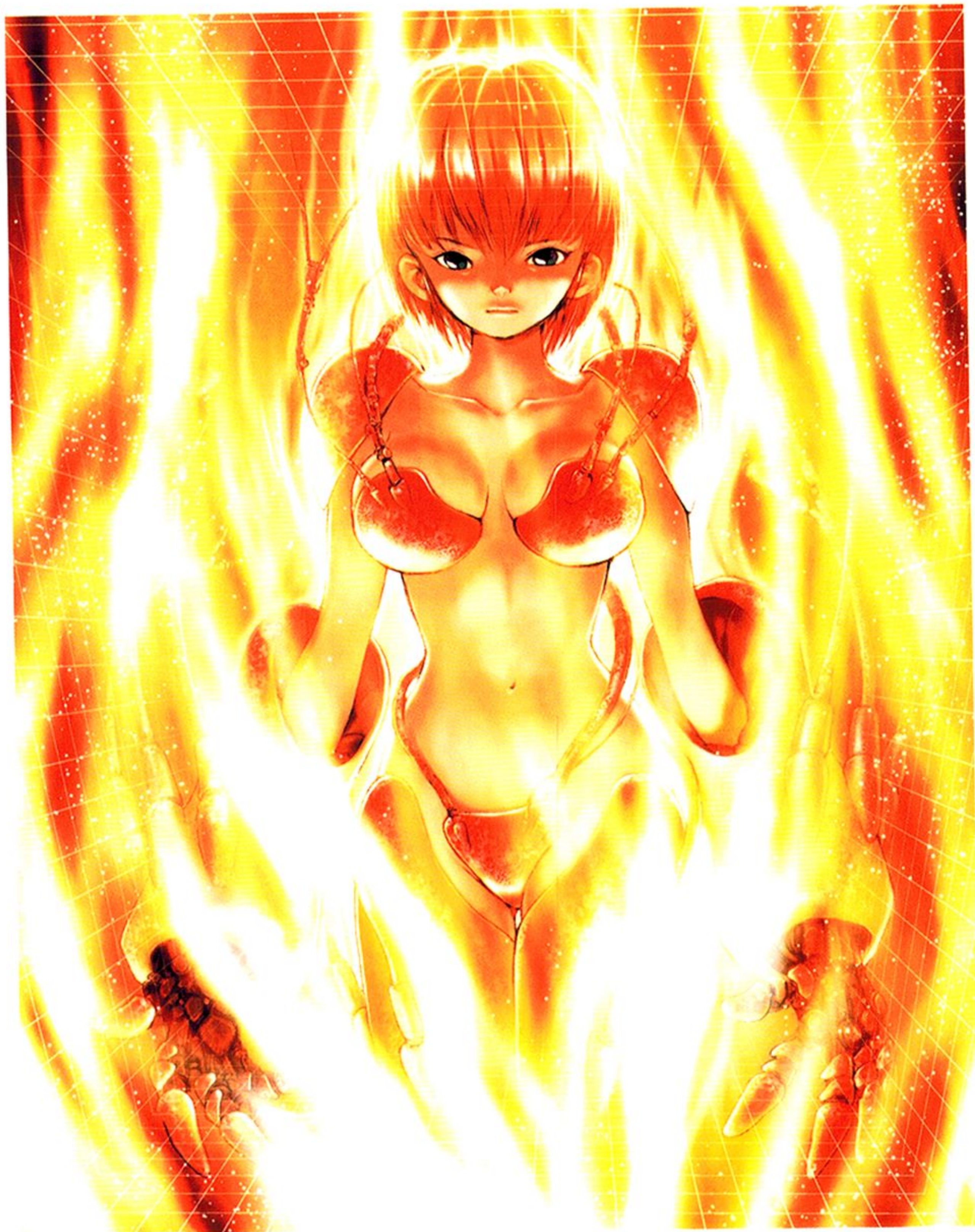
格子データを作ってみます。まず，こんなデータを作ります。

grid1.bmpを繰り返しコピーで並べます。LightEDitorでは，メニューからREPEAT COPYを使います。

grid2.bmpのアルファデータとして，こんなグラデーションデータを用意します。LightEDitorでは，ドラフターパネルのGRABOXで全体を塗りつぶします。

LightEDitorのパース機能を使ってみます。こんな感じで設定してみます。

パースのついた箱を人物を囲むように設定します。

パースのついた箱の各面に格子のテクスチャを貼り付けるといった感じの設定です。変形コピーを各面に対して行うわけです。

ブラシパネルで「コピー明alp」筆を選びます。この筆はコピー筆の拡張で，コピー元のアルファデータを参照して濃度値とします。「あかるく筆」と同じ計算式なので，格子の黒い部分はコピーされず，白い格子部分だけがコピーされるように見えます。

さらにグラデーションの濃度が影響されるので，徐々に消えていく感じでコピーされます。

すが，自分でそのように作ったものなのでよしとしましょう。

余談ですが，変形コピー機能をLightEDitorに組み込むとき，変形先のx，yの座標から変形元の座標を算出するために，どうしたらいいか苦労しました。文献とか見つからず，知ってる人もいなかったので，変形コピーの方法については完全に我流です。それでも，考え方についてはあってると思いますので簡単に紹介しておきましょう。

基本的には三角関数でコピー先の座標から，コピー元の座標を探し当てればいいわけですが，「比例関係にある2つの直角三角形」をどこに探して，どうやって比率を求めるかで困りました。

変形先の四角形の周りに直角三角形を探してもうまく見つからず，方程式を組めなかったのです。

「変形された四角形」というのは，「パースのかかった長方形」だと気がつきました。変形された四角形は，対になる辺を延長すれば，消失点が必ず2つ求められます。それを基準にすれば，明快に直角三角形が探せます。同時に，コピー元の座標も簡単に得られることがわかりました。

変形先四角形の各4隅の点から，消失点A，Bを求めます。A，Bは水平であるとは限りませんが，わかりやすく水平であるとしてこれ以後の座標を計算します。

Dを基点に平行線を引き，辺を延長させて，C，Eを求めます。消失点から，x，yを通り線分D，Eと交わる点x'，y'を求めます。CD，DEは，コピー元の四角形の辺の比に相当しますので，すぐにコピー元座標の比が求まります。

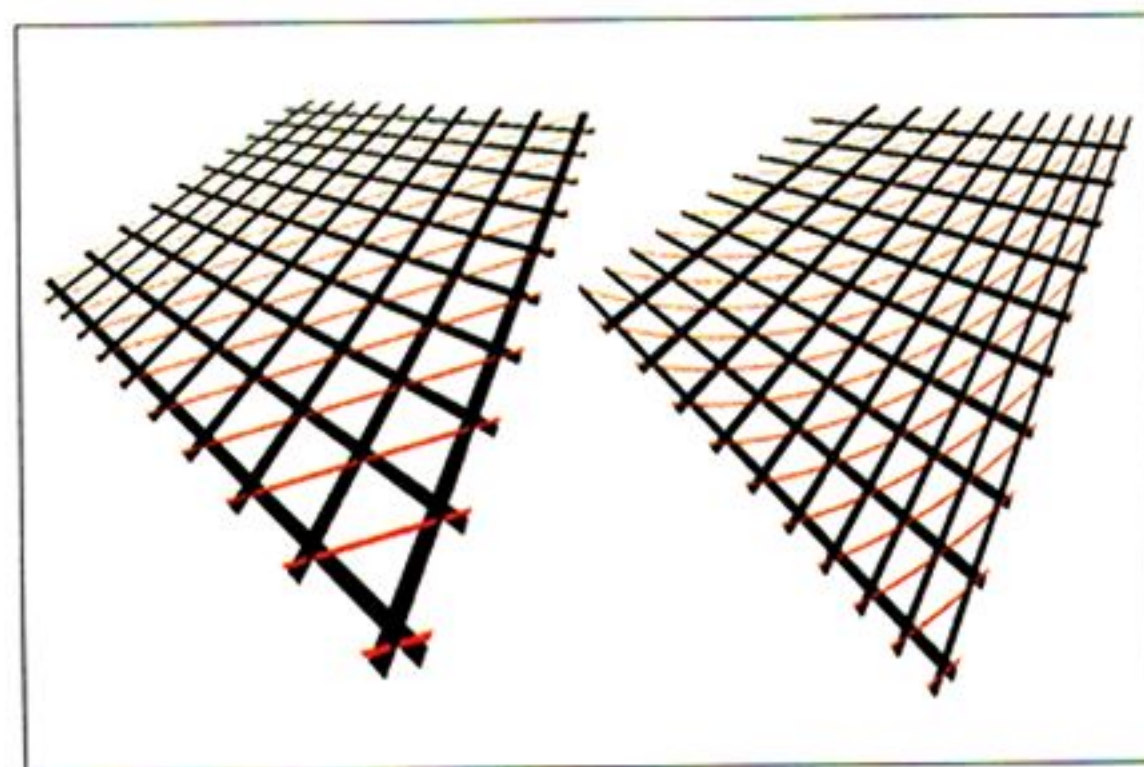
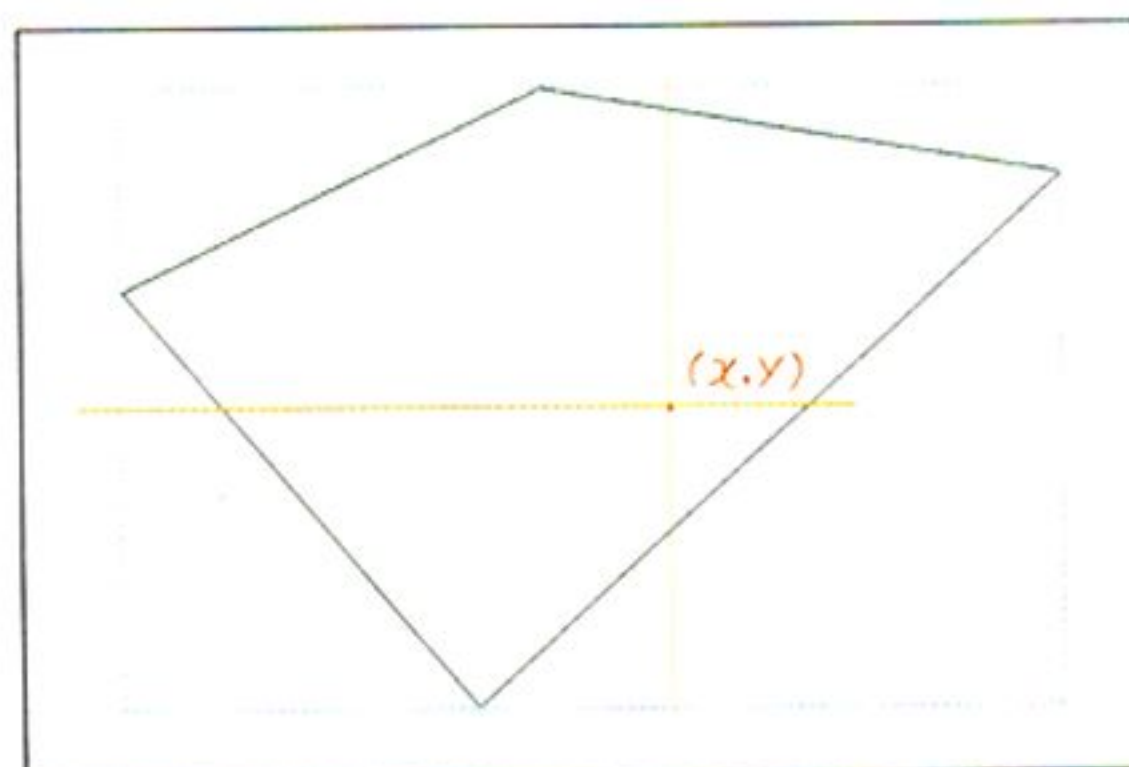
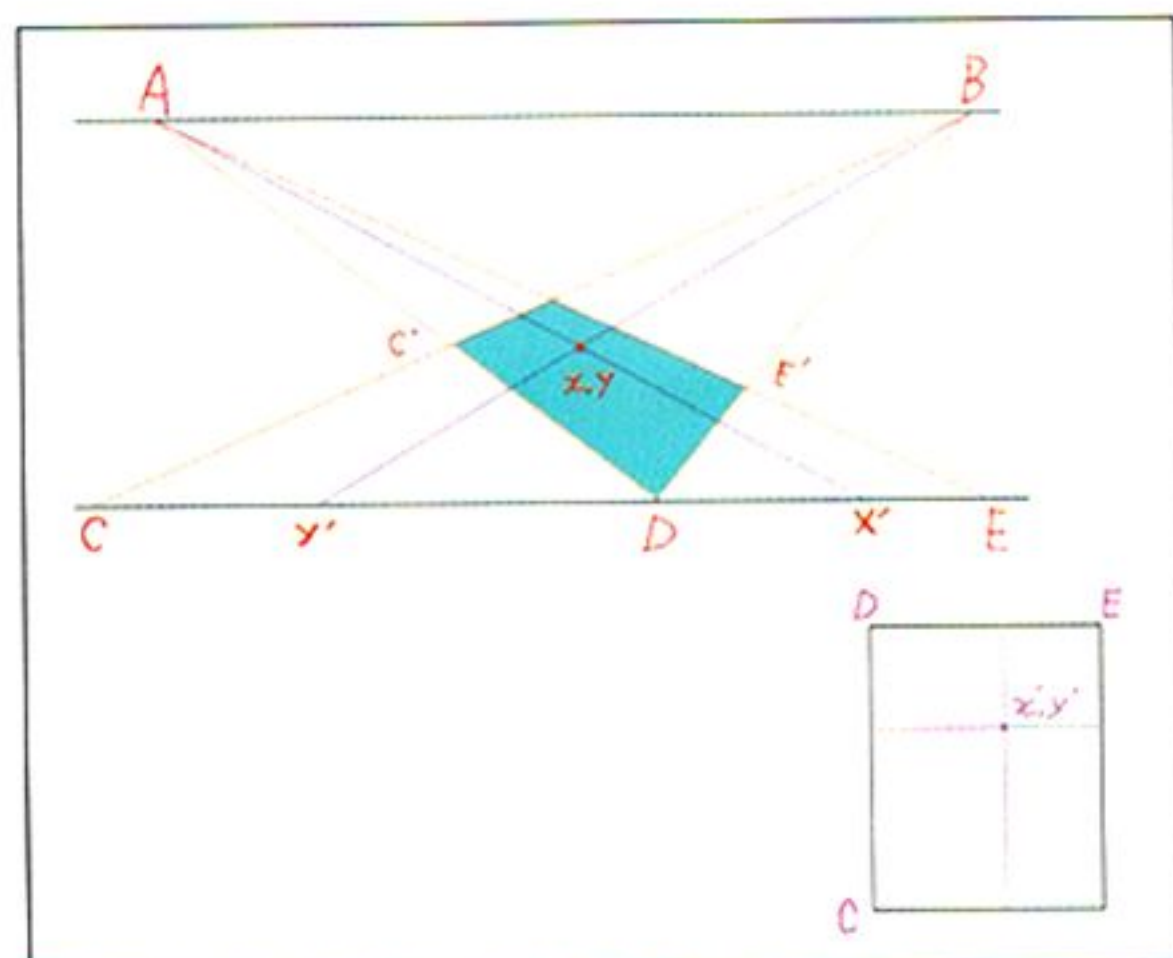
パースを基にしてるので，ビルの壁に絵を貼り付ける場合でも，正しく変形が計算されます。

これを応用して，消失点に向かう線分が簡単に描けるようなパース機能も組み込むことができました。

ちなみに，C' D，DE' から，比を求めると，平面上でよじれたような，パースのかからない変形コピーになります（右図）。

最後に，定番の「きらきら」を加えて完成です。

ドラフターパネルで，grid2.bmp全体を範囲指定して，コピー元に設定します。

あとは「実行」ボタンひとつで，サイバーな格子がふちに現れました。人物に向かって消えていってくれています。モロにツールの機能にたよってま

ポートレートっぽく描く

川原由唯　Kawahara Youi

今回は媚びてニコパチしてみました。大口径望遠レンズによるポートレート写真を意識してます
（シチュエーションは，２年くらい前に描いたもののリメイクですが，絵はまったくの新作）。

ボケ味の話

雑誌のアイドルグラビア写真の背景に注目してみてください。なにが写っているのかわからないほど，ボケボケにぼかしたものが多いことに気づくと思います。これは鑑賞者の視線が被写体となる人物に自然に向かうように，写真家が意図的に被写体以外をボカしているのです。写真の素人さんは「ピントが画面全体にいき届いた（被写界深度が深い）写真がうまい写真」と思いがちですが，鑑賞して楽しむための“作品としての写真”では被写界深度の深い写真は必ずしも正解ではありません。グラビアのポートレートでは，いかに滑らかに背景をボカし，主役を浮き立たせるかがカメラマンの腕の見せどころなのです。

写真を多少なりともかじったことのある方ならご存じかとは思いますが，ボケの量は，被写体のコントラストやフォーカス面からの距離（前後どちらに振れるかによっても異なる），絞り値などが大きく影響します。また，「ボケ味」はレンズの光学設計に依存しており，焦点距離，解像度，コントラストなどと並んで，カメラマンがレンズを選択するうえでの重要な性能のひとつになっています。

ボケ味はどちらかというと撮影者，鑑賞者の好みに依存する感性で測るべきパラメータで，数値化できるような特性ではないのですが，一般には高価なレンズほどボケ味も考慮されて設計されているといわれています（単焦点の大口径レンズほどボケの足が長く，滑らかなグラデーションがかかる。ズームレンズは光学系の設計に無理があるため，うるさい「二線ボケ」の発生など，一般にボケが硬く汚い）。

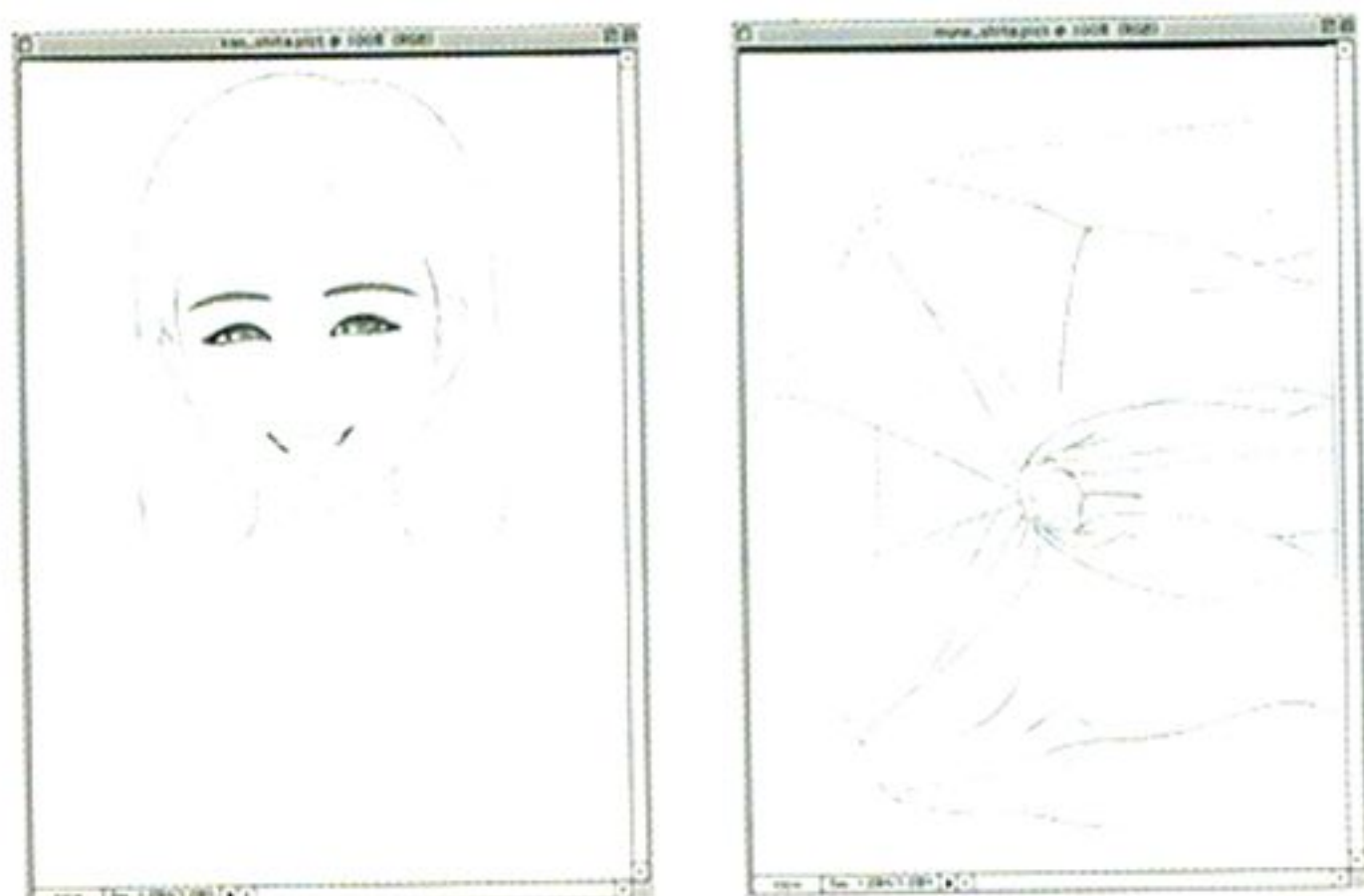
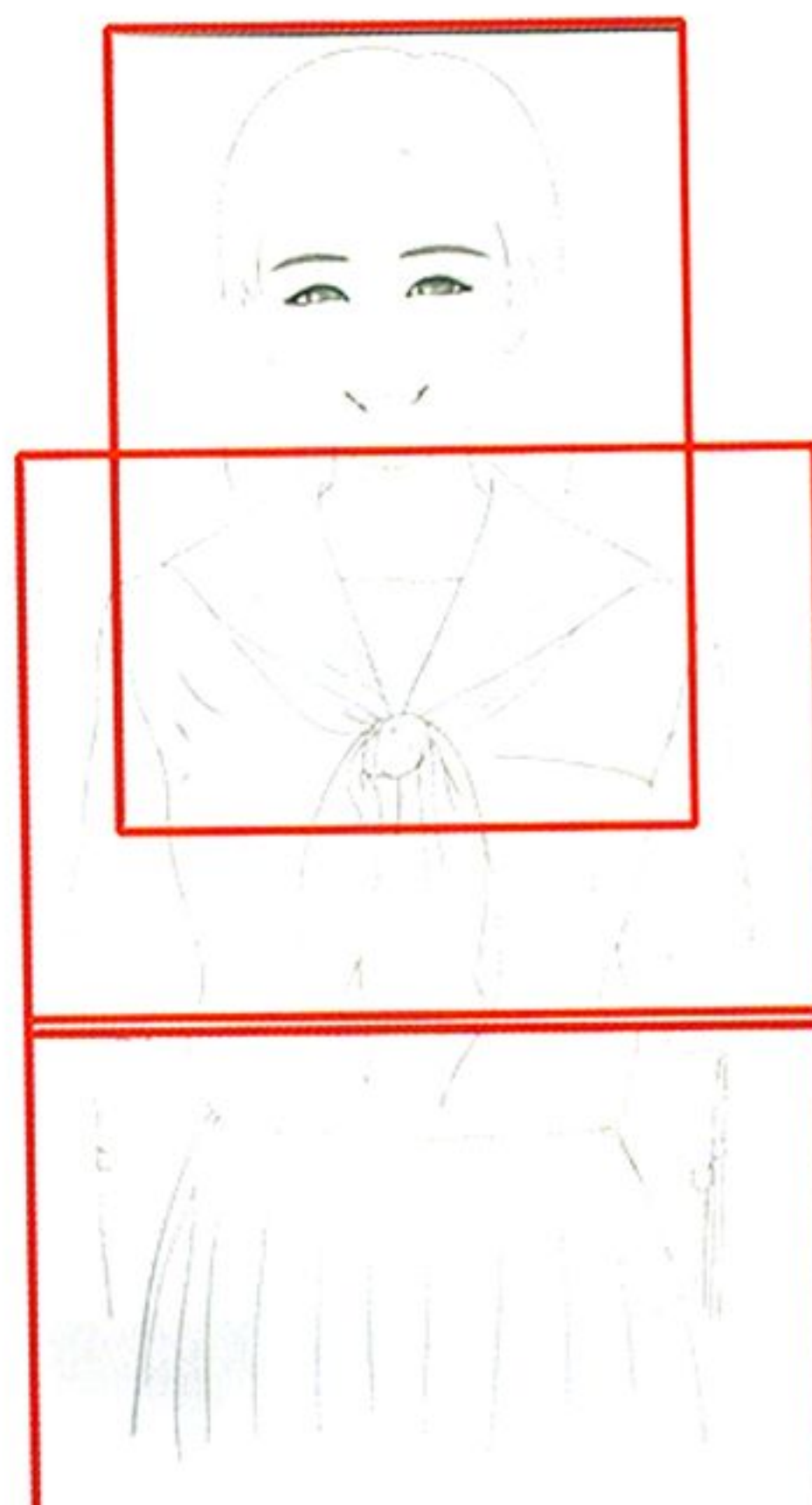
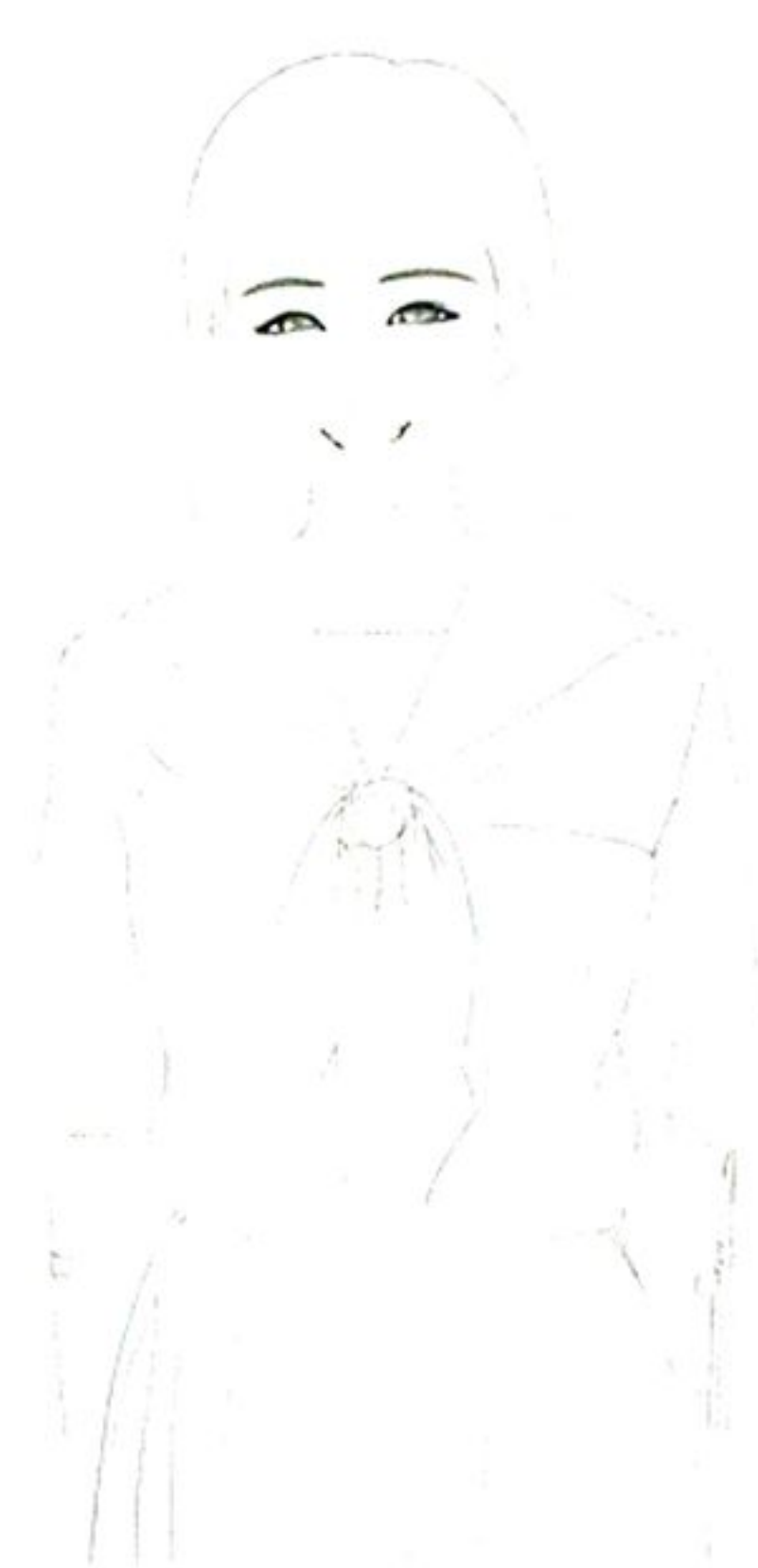
まぁはっきりいっちゃうと，写真オタクのウンチクレベルの話なんですが，いずれにせよ「ボケ」は写真の美しさを語るときに，とても重要な要素になることを覚えておくとよいと思います。写真の見方が変わること必至です。ボケを美しくすることに主眼を置いて設計されたレンズさえ存在するくらいですから……。

というわけで，今回はボケを生かした作画をしてみました。

マシン環境

お絵描き環境は変更まったくなし。G3Macほしいけど，先立つものも置き場もないです。

本体 Power Macintosh 7500/100
アクセラレータ MACh SPEED 604e 233MHz
搭載メモリ 352MByte
VRAM 4Mバイトに増設
ハードディスク　内蔵1G＋4Gバイト
ディスプレイ 17 inch トリニトロン（調子悪い）
タブレット WACOM UD-608 II ADB
フラッドベッドスキャナ JX-250
230M のMO，CD-R，フィルムスキャナなど
MacOS 8.1
Photoshop 4.0.1J

下描き

いつも通りです。スミマセン。これを描きはじめた頃，True Love Story2（通称下校煮）っつーギャルゲーに凝っていたので，セーラー服の女の子です。A4のケント紙に主役の子をシャープペンシルで下描きします。描いているうちに1枚では収まりきれなくなってしまったので（これも予定どおり），紙を継ぎ足して加筆。縦に使うと肩幅が狭すぎるので，肩から下は紙を横に使いました。最終的にA4×3枚で完成。

スキャナで下描きを1枚ずつ取り込んで，Photoshop上で位置をあわせて1枚の画像データにします。

トーンカーブやコントラスト補正をかけて線画をクリーンナップ。紙の継ぎ目は線が途切れていたり，濃淡が汚くなっているので，鉛筆ツールや消しゴムツールを使って不自然な感じがなくなるまで修正。

着色

春号で解説した手順にて，線画をセル画レイヤー化。あとはいつもどおり，肌色，制服，髪の毛，スカート，スカーフをそれぞれ別レイヤーにして，塗り絵をします。

濃度の高い色で陰影を入れます。光の方向を考えて（実はあんまり考えずに習慣的に入れてるんだけど（笑）困ったもんだ）覆い焼きツールなどでハイライトを入れていきます。照り返しや透けてくる光を考えて，陰に肌色を薄く入れておくと深み（？）が出ます。水彩と同じ要領です。

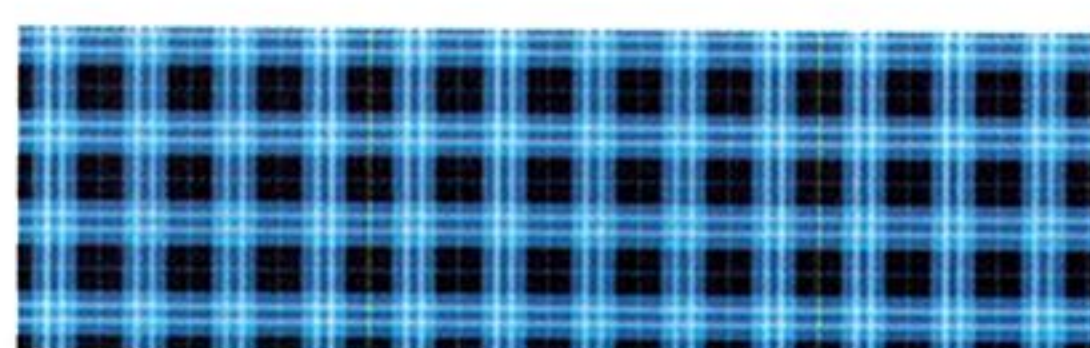
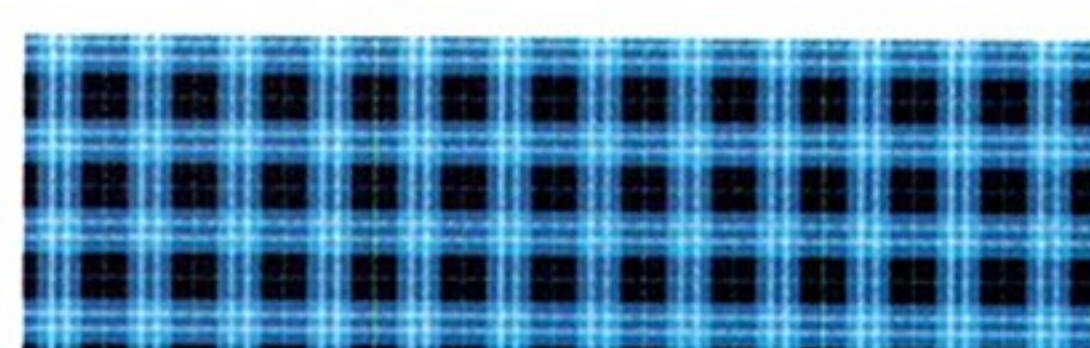
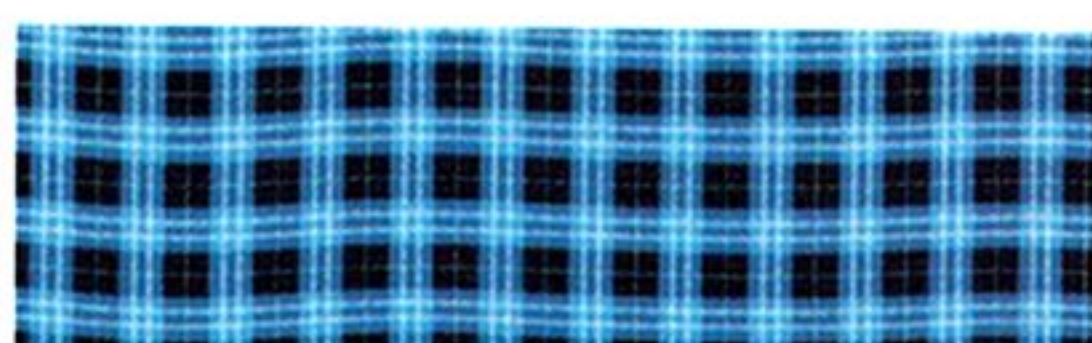
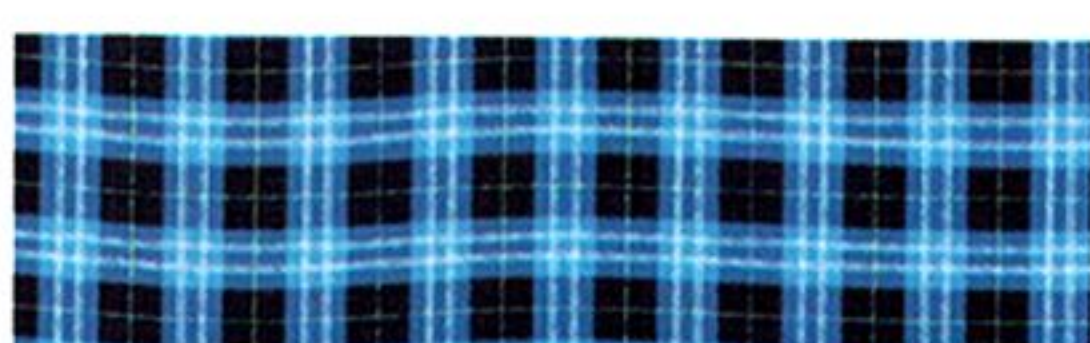
陰影を整えて，下地が完成したところ。

スカートの影を入れたり，顔のディテールを書き込みます。

髪の毛など，細かい部分を整えます。かばんのハイライトを入れ，これでほぼ完成。スカーフの色は青に変更。

チェックのスカートを描く

スカートはチェック地にします。最初はスカートのヒダにあわせてパターンをカットし，貼り合わせていこうとしたのですが，角度の調整が難しく，2，3枚貼ってみたところで労力の割に存在感が出ないことがわかったのでやり方を変えました。

まずチェックのパターンを自作します。矩形領域に塗りつぶしたレイヤーを何枚か重ねて表現すると簡単です。

先の画像を横に連結，スカートの正面部分全体になる程度の大きさ，縦5，横10程度に延長します。

「波紋（小）」のフィルタで，少々画像を揺らがせます。これで糸目の雰囲気になります。最終的にはほとんど見えませんけど。

「波形」フィルタの上下パラメータのみ反映させて，波打たせます。

左右方向が長すぎそうだったので，若干カット。

「球面」フィルタを横方向のみに反映させて，円柱状に変形。

「シアー」で弓状に変形。

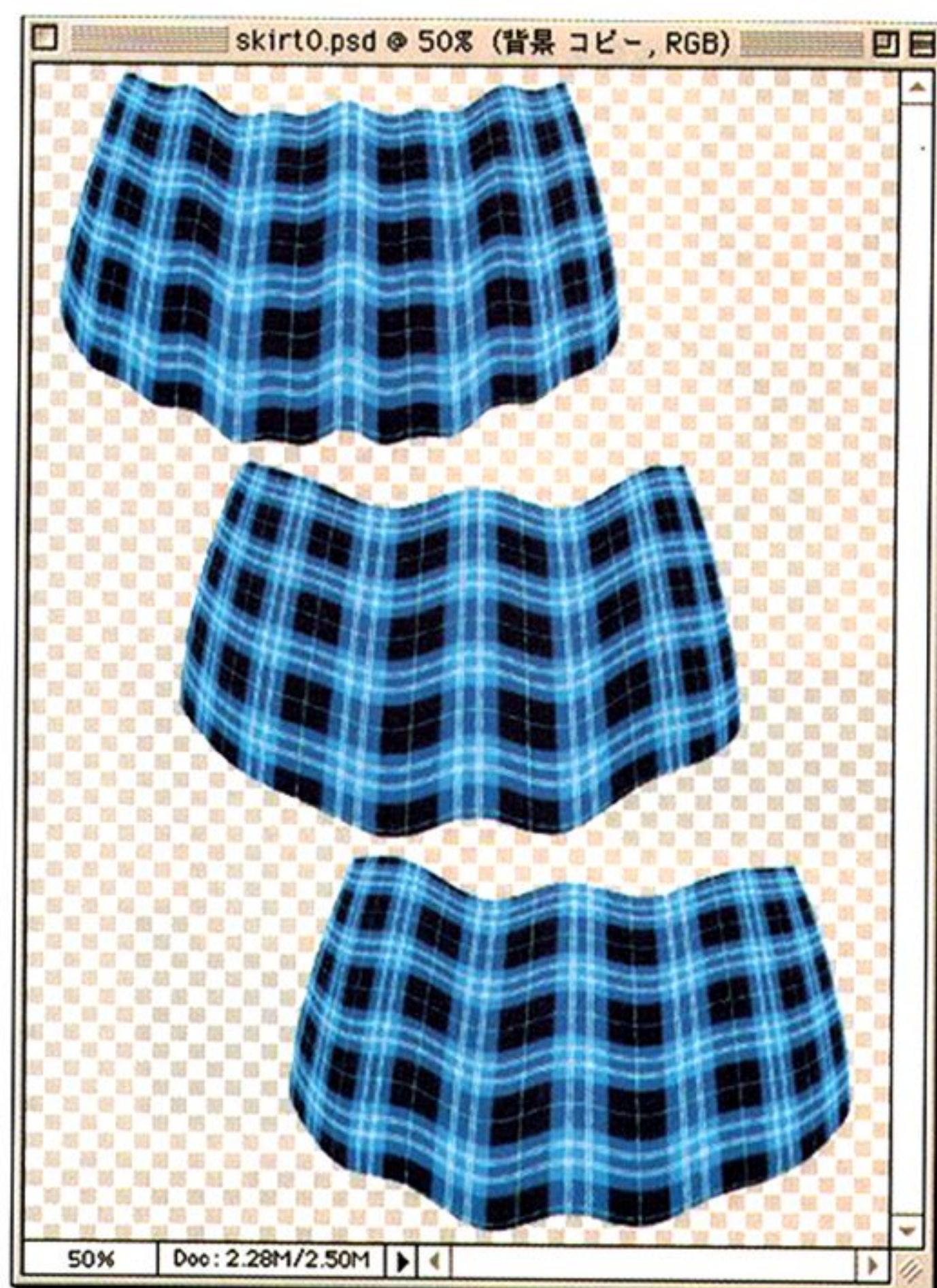
今度は「球面」フィルタを両方向に反映させて，球状に変形。

「波形」フィルタを上下方向に反映させて，波打つ布を表現。パラメータをランダムにして，形状を数パターン用意しておきます（今回は図のとおり3パターン）。これをそれぞれ別レイヤーにし，ヒダにあわせてカットして重ねることで自然な感じを表現します。

skirt0.psd @ 50% (背景 コピー, RGB)

50% Doc: 2.28M/2.50M

OhIX3_制服少女04.psd @ 50% (布下地4, RGB)

50% Doc: 2.20M/7.96M

スカートのパーツは，こんな感じで切り分けている。3枚のレイヤーを線画に合わせてカットし，チェック模様の適度な「ずれ」を表現。

スカートを完成させる

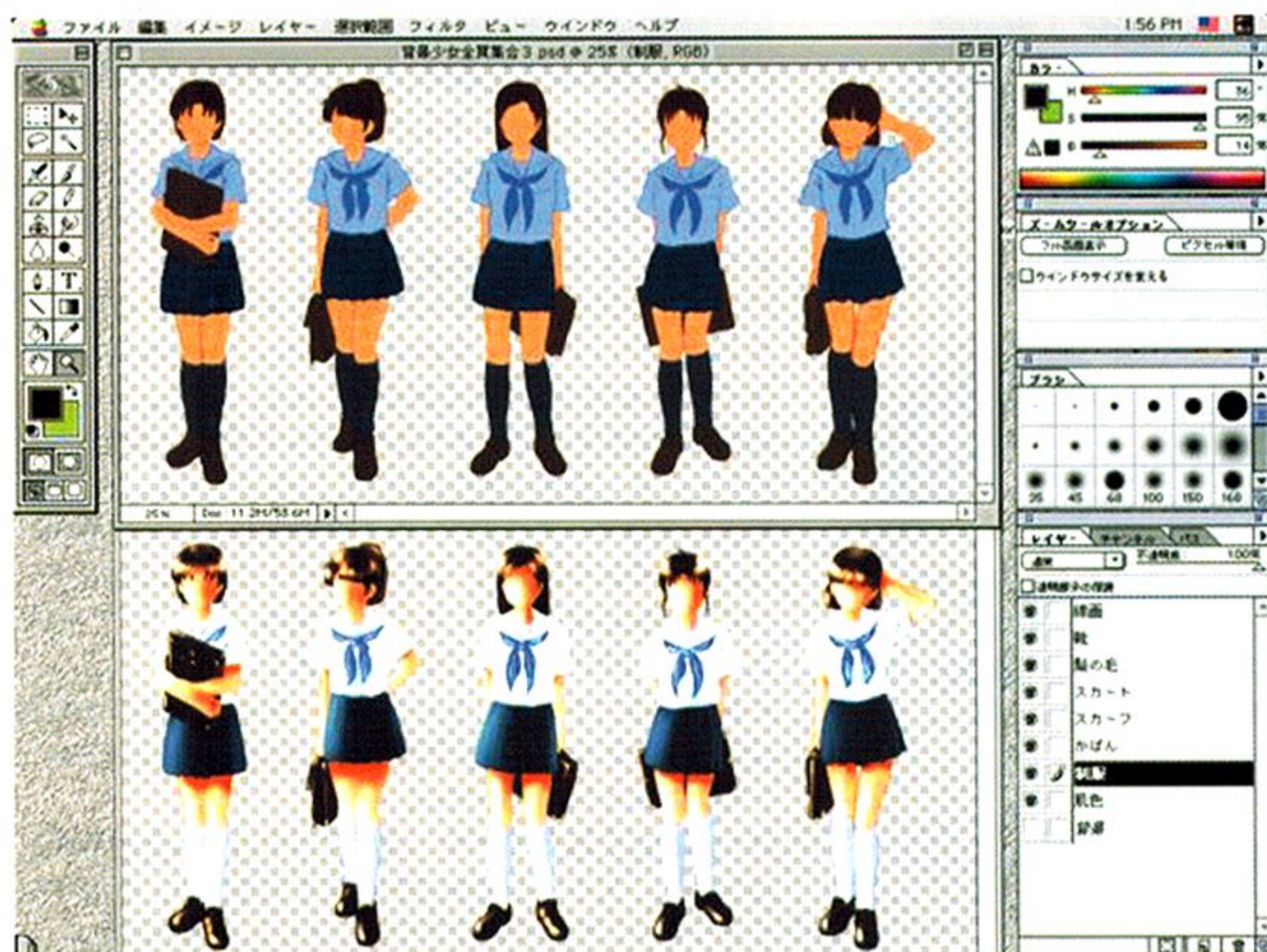
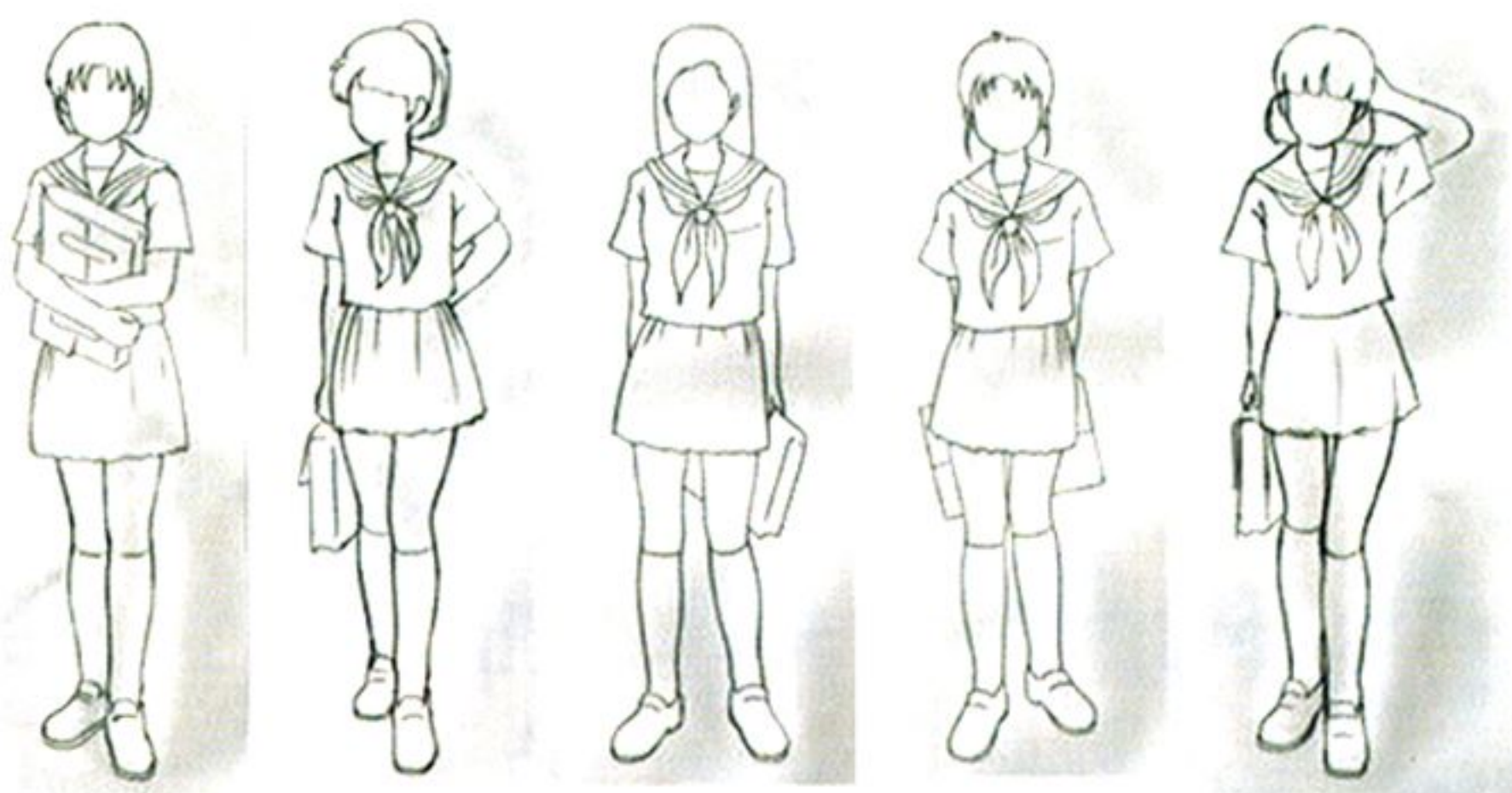
スカートのパーツは，こんな感じで切り分けている。3枚のレイヤーを線画に合わせてカットし，チェック模様の適度な「ずれ」を表現。

パーツを貼り合わせたところ。まだ陰影に乏しいので立体感が足りません。

とりあえず主役の完成

もう少しがんばって，色を整えて，主役は終わり。うぅ，色入れたらあんまり可愛くなくなっちゃったよ，この娘（涙）。

脇役を描く

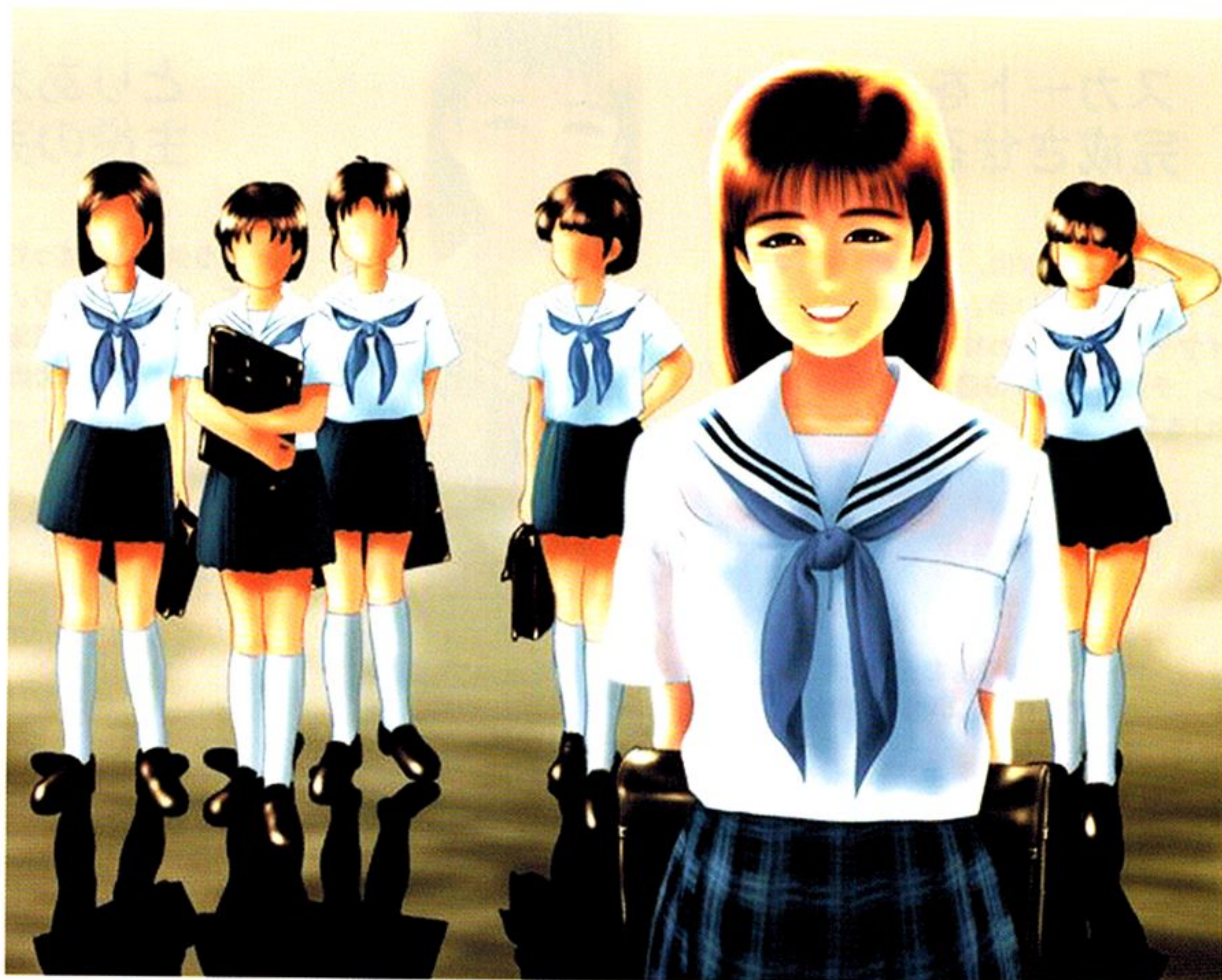
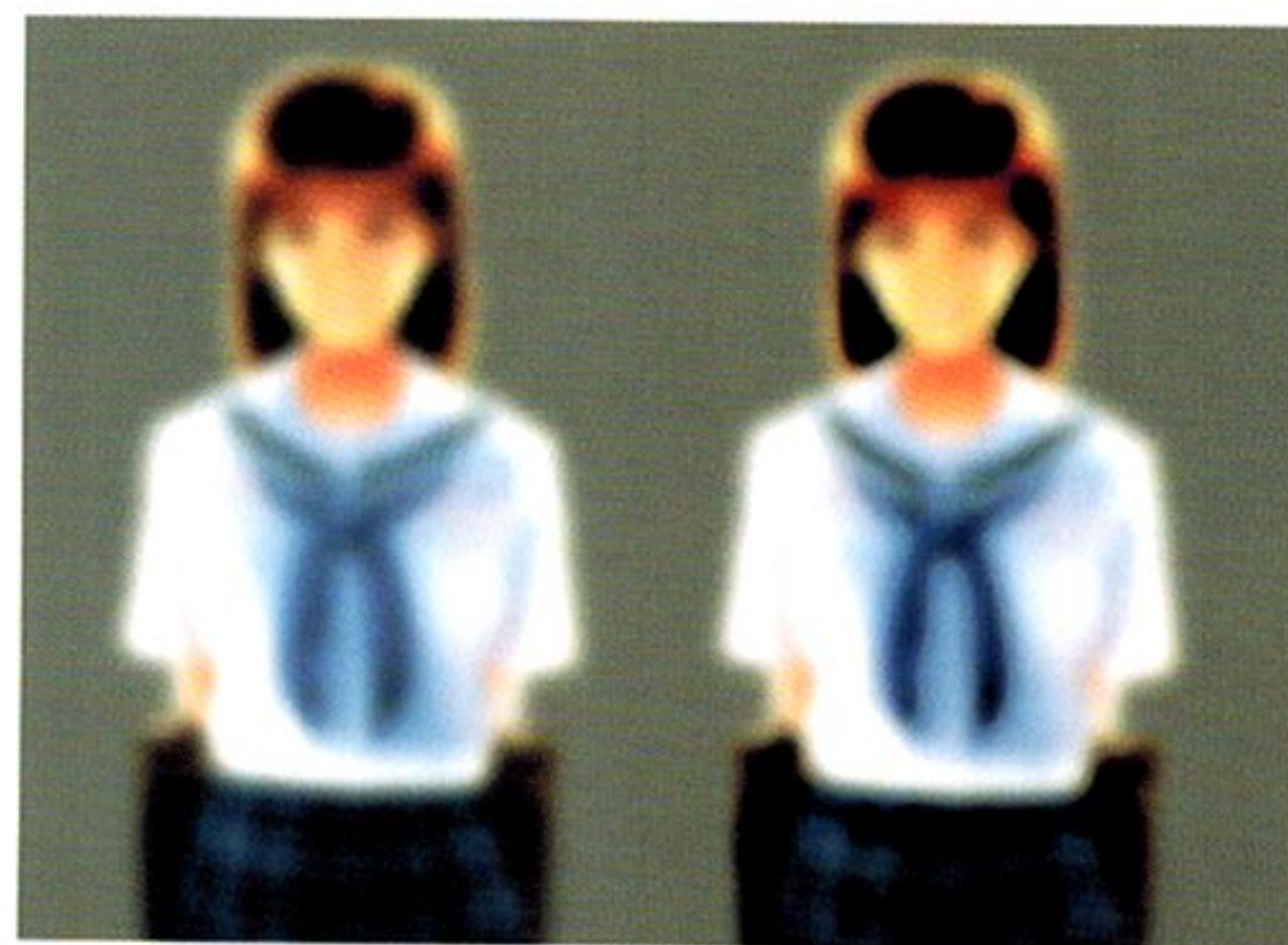
後ろに立っている女の子たちを描きます。
最終的にボカしてしまうとはいっても，あまりいいかげんに描くと結構わかってしまうもんです。手を抜けるところは抜いて効率化を図るべきですが，ていねいに仕上げたほうが結果がいいのは当たり前。

輪郭の下描きです。デッサン力のなさが露呈しています（涙）。
顔やスカートのチェック模様などはボカせば潰れてしまうのは明らかなので省略します。レポート用紙の裏なので紙のシワや罫線が映っちゃってますけど，適当に修正できるのでよしとします。

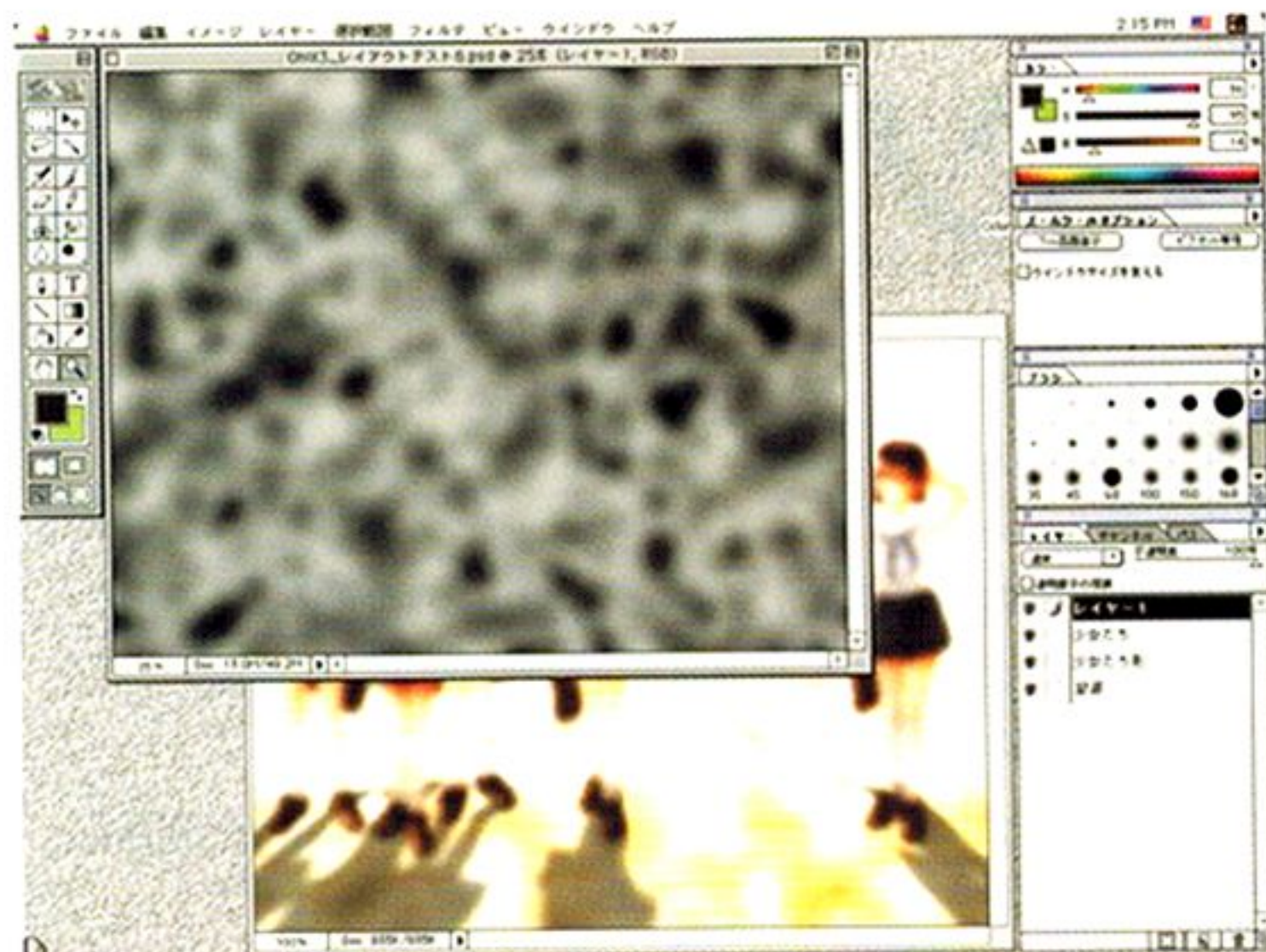
コントラスト補正でクリーンアップしたあと，セル画化しておきます。

ひたすら塗ります。塗りムラや塗り残しがあると，ボカしたときにも意外とwかるので，あまり手を抜かないほうがいいです。
ルーズソックスじゃないのは嫌いだからではありません（別に嫌いじゃないです）。

覆い焼きツールやコントラスト補正で陰影をつけます。ソックスの色は当初は黒だったのを白に変更してます（説明手順的に前後してるけど，実はボカすと汚いことがわかったので）。

レイアウトと仕上げ

背景用の女の子たちをそれぞれ別のレイヤーにし，主役の子の後ろに適当に配置してみます。適当といってももちろんパースペクティブを考えて，水平線（HL）を意識してますが。この絵の場合は目のあたりが水平線になってます（そういうつもり）。
女の子たちの影は，女の子たちのレイヤーをコピー，上下反転して位置を移動し，色調補正の彩度明度変更で黒くしています。影の形状は正確ではないですが（〝足〟のところなど），どうせボカしてしまうつもりなのでこれでおっけー。

女の子たちと影を「ぼかし（ガウス）」フィルタでボカします。もうこのあたりのボカし具合とか，色調整はほとんど試行錯誤なので，記憶に残っていません。影は，レイヤーの変形をしてパースをつけていますが，レンズの焦点距離が長いことを考えると，実際にはこんなに極端に遠近感つかないと思う。でも雰囲気なのでこれでおっけー。

２線ボケをシミュレートしてみるのもいいかもしれません。本当はそんなところに創作パワーを割くくらいなら，絵の題材（核心）のほうに力を入れたほうがいいのはいうまでもありませんが参考までに。
ペイントソフトの一般的なボカシフィルタは，周辺のピクセルに対して単純に距離に反比例して色を加算しています（もっと凝ったのもあるかもしれないけど）。つまり理想的なボケ方をしてしまうため，CGで単純にボカシをかけても，よく見ると写真のボケとは異なります。
写真の場合は，光の回折，レンズの収差によってボケにもムラや滲みが発生し，特に目立つところでは輪郭に沿って不連続な線が発生します（ボケのエッジが立つ，などという）。これを，２線ボケといいます。
２線ボケはガチャガチャとうるさい感じになるので普通は嫌われますが，それが写真の味になる場合もあるので，一概に悪いとはいえません。
ほとんど自己満足ですが，次のようにしてみると雰囲気出るかも。もちろん絵的にそう見せるだけで，光学をシミュレートしているわけではありません。念のため。

対象となるレイヤーを２枚にコピーします。
そのうち片方に「光彩拡散」フィルタをかけます。
ボカします。
アンシャープフィルタで輪郭を少し立てます。
もう一方のレイヤーもボカします（ボカし強度を変えたほうが いいかもしれません）。
下になるレイヤーを「通常」で，その上に重ねるレイヤーを「比較（暗）」で重ねます。そうすると，適度にボケ足にムラが発生して，二線ボケっぽくなります。
単純にボカした背景と，２線ボケっぽく加工した背景を比較してみると，なんとなく２線ボケのほうが写真っぽく見えると思います……？　うーん，やっぱ自己満足かなぁ。

雲フィルタで描いた模様にボカしをかけて，「覆い焼き」モードで背景の女の子たちに重ねます。適度に揺らいで，逆光が滲んだ感じがでます。

いわゆるゴーストを描きます。ベタ塗りしたレイヤー上に「逆光フィルタ」を使って描いたゴーストをコントラストを強く補正しておき，レイヤーオプションを「スクリーン」にして，90パーセント程度の濃度にして重ねます。
Photoshop の「逆光フィルタ」って誰が使っても同じ効果になるから嫌われるみたいだけど，こういうところに使うのならいいよね？

ゴーストを重ねた結果。まぶしいでしょ？

背景が暗い絵の場合は，覆い焼きツールを使って円形に滲んだ光を背景に散らすといい雰囲気になりますが（僕はときどきやる），今回の絵は背景全体が明るいので入れてません。
こういったボケたキラキラは，実際の写真では口径食の効果がでます。でもさすがにCGでそこまで凝っても絵的に面白いとは思えないですね。雰囲気的にはゴースト像の外側を薄くすることで描画できるかもしれません。
口径食というのは，画面の周辺部の光線（軸外の光束）がレンズの枠などによって遮断されて周辺部の光量が低下する現象で，もちろん発生させないに越したことありません。ゴーストの形状が，画面の中央から外れるにしたがって円状からラグビーボール状に欠けていくことでわかります。絞りを絞り込むとなくなります。

最後に

ペイントソフトでは簡単にボケを表現できる割には，巷にあふれるプロアマ2DCG作家さんのなかにボケを使う人って少なくないですか？　僕が鑑賞している対象が漫画系のCGに偏っているからかもしれませんけど，なんか不思議なんですよね。

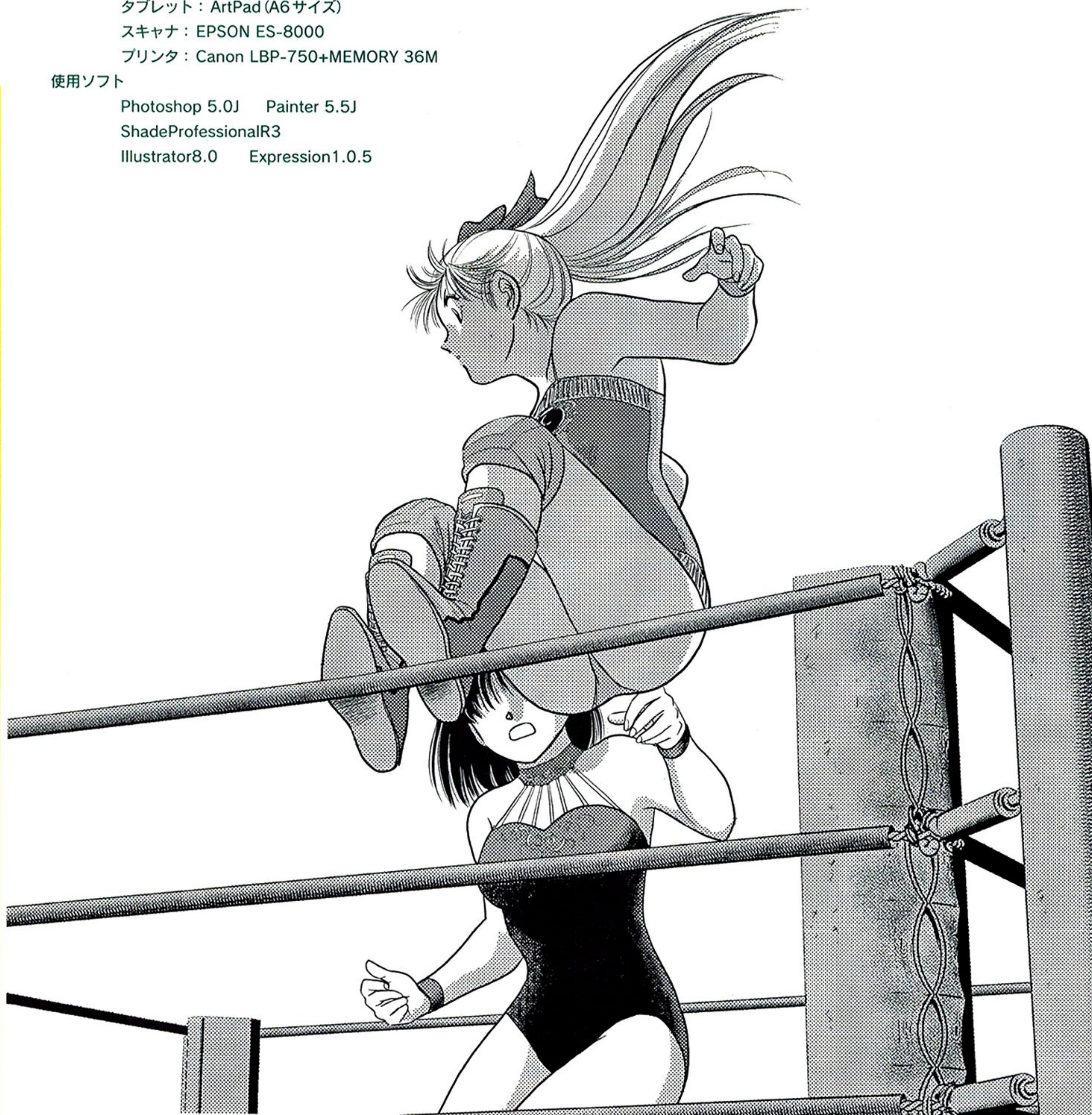
もっとも，安易にボケを使って絵を台なしにしているケースもときどき見られます。絵そのものは凄く上手いのにボケが不自然なため存在感を壊している作品があります。ある有名なプロの漫画家さん（比較的古くからMacでCGやってた人）が描いた絵なんですが，パースペクティブ的には標準～広角焦点距離のレンズで切り取った画面なのに，遠景と近景が大きくボケているんです。ティルトレンズといった特殊なレンズでも使わないかぎり，普通の写真ではこうはならないために，なにか模型のジオラマを見ているような違和感を感じました。おそらくその作家さんが，ボケがどういうケースで発生するのか知らないため，ただなんとなくボカしちゃったんだと思うのですが。

遠近感を出すためにナンでもカンでもボカしてしまおうというのは危険です。漫画絵とはいえ，どんな焦点距離のどんなレンズでどんな被写体をどう切り出せば，どんな絵になるのかというようなことが無意識に意識できるようになるために，写真や光学の初歩的な知識はあって損はないなぁと思います。

パソコンによるマンガ原稿の作り方

森川久志　Morikawa Hisashi

使用環境：
- PowerBookG3/292 memory192Mb
- タブレット：ArtPad（A6サイズ）
- スキャナ：EPSON ES-8000
- プリンタ：Canon LBP-750+MEMORY 36M

使用ソフト
- Photoshop 5.0J　Painter 5.5J
- ShadeProfessionalR3
- Illustrator8.0　Expression1.0.5

600dpiレーザープリンタの出力を原寸で掲載

クッ

！

お馬鹿さん！

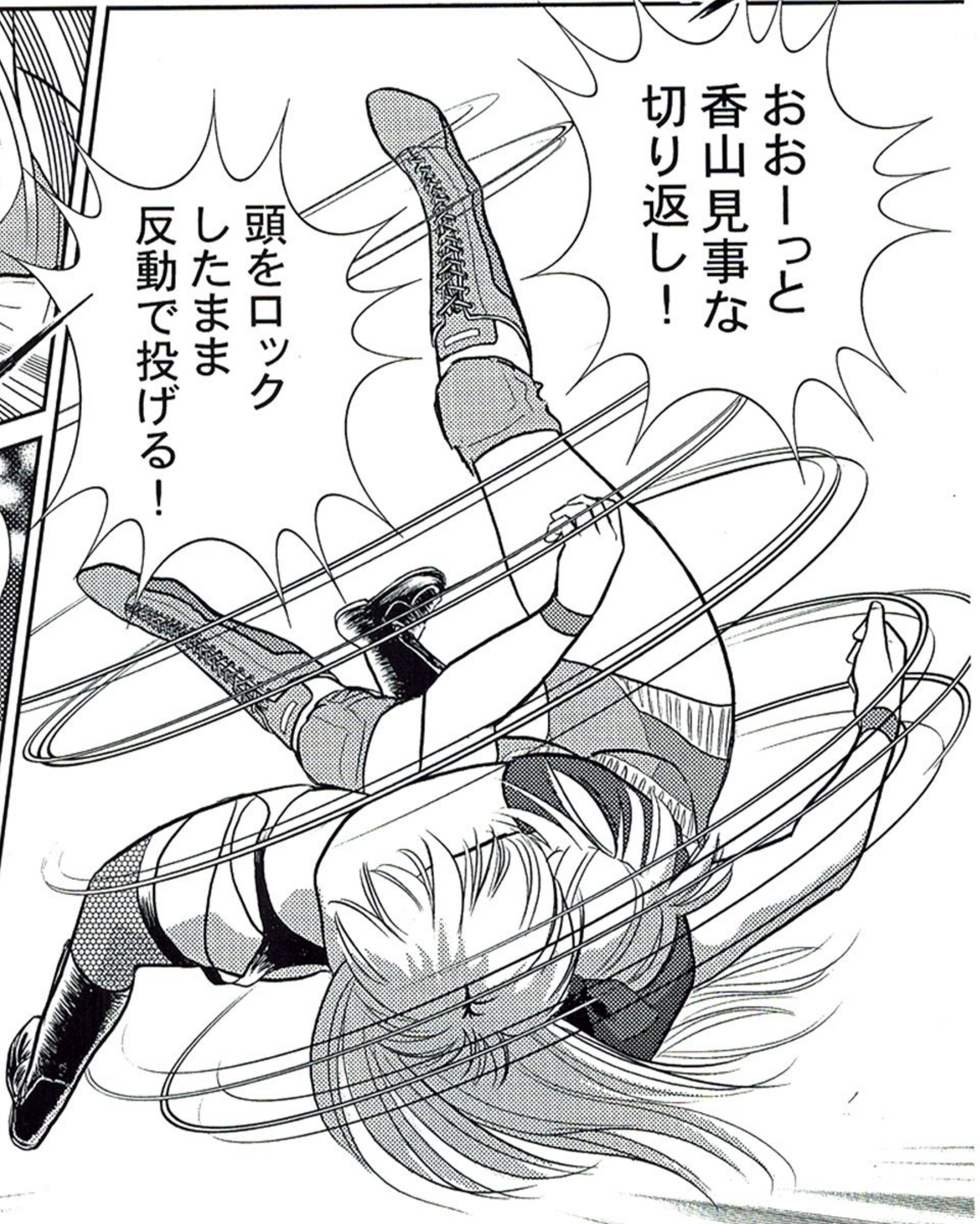
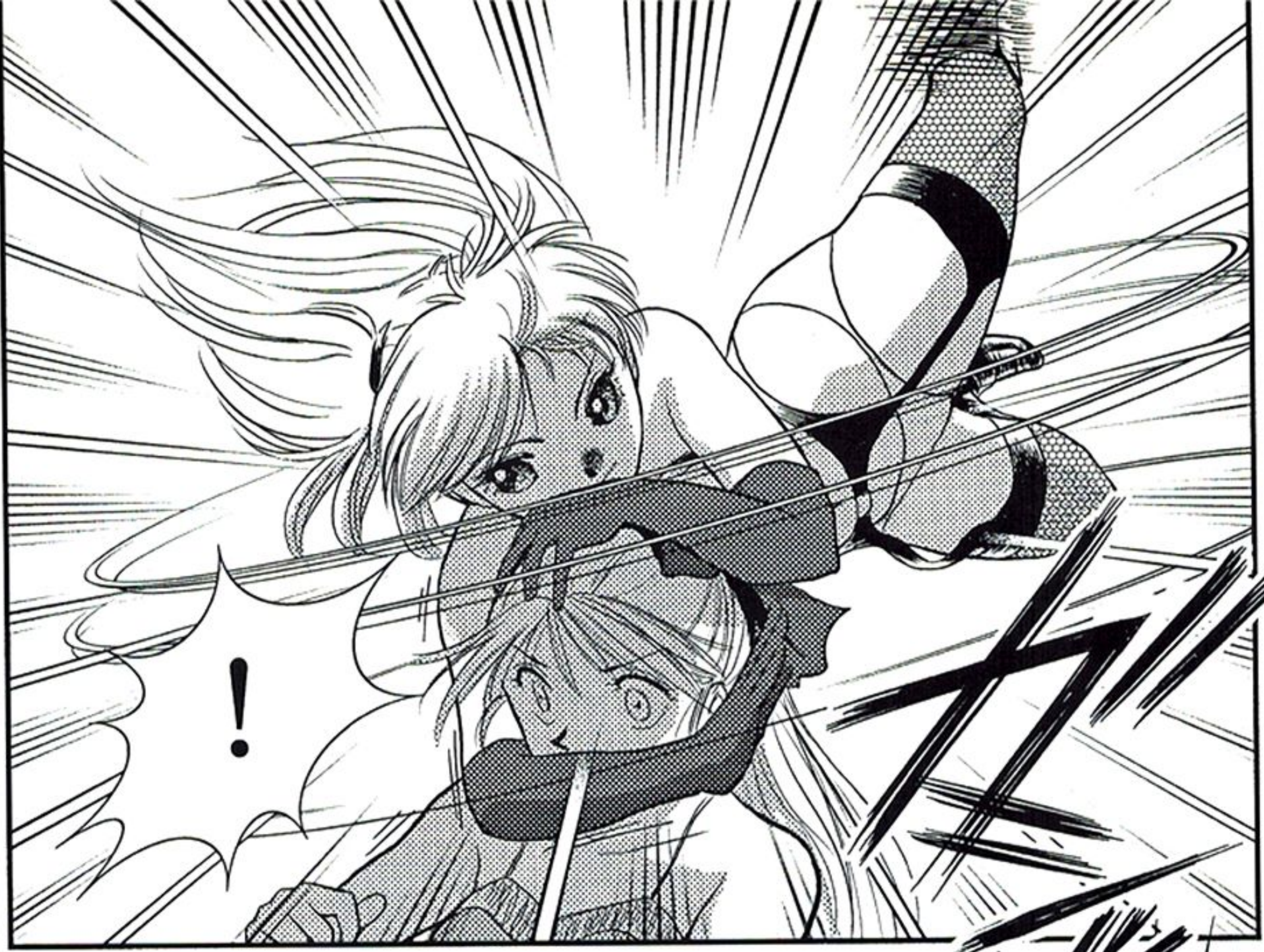
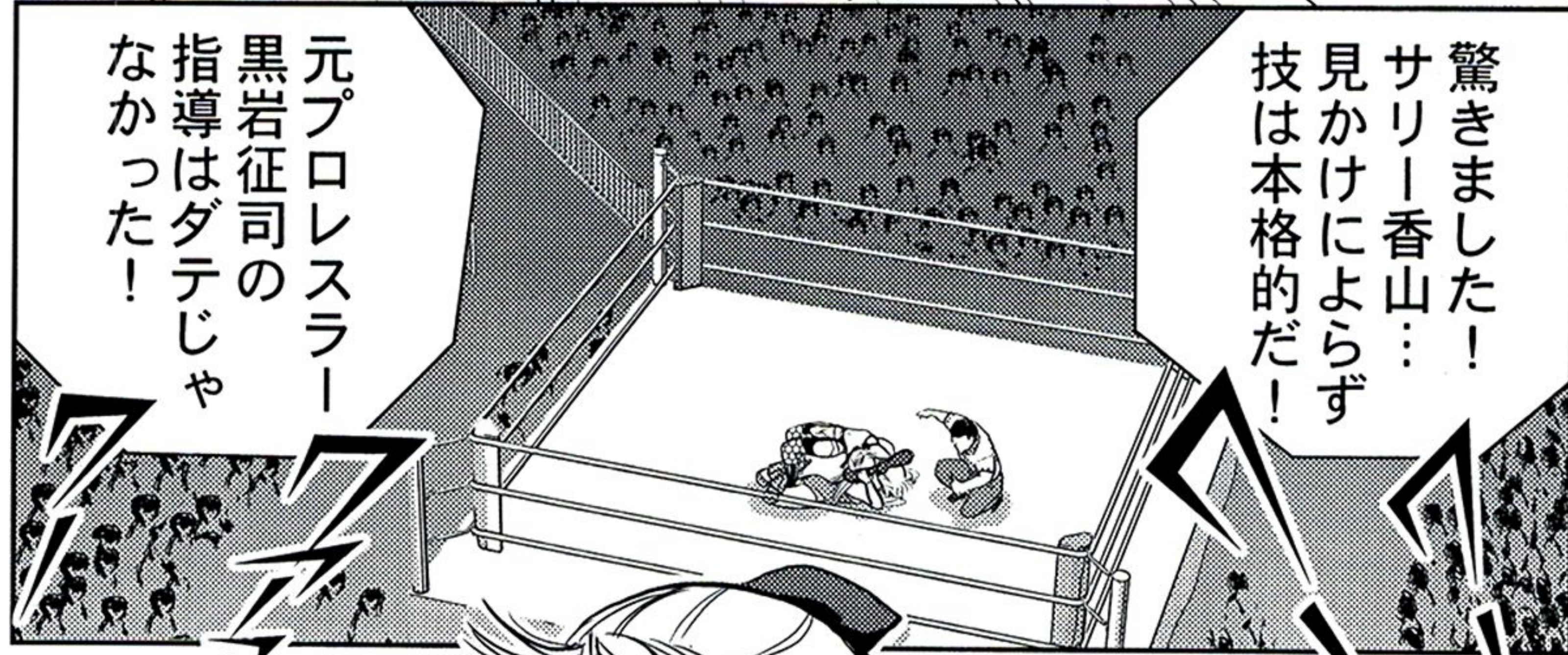
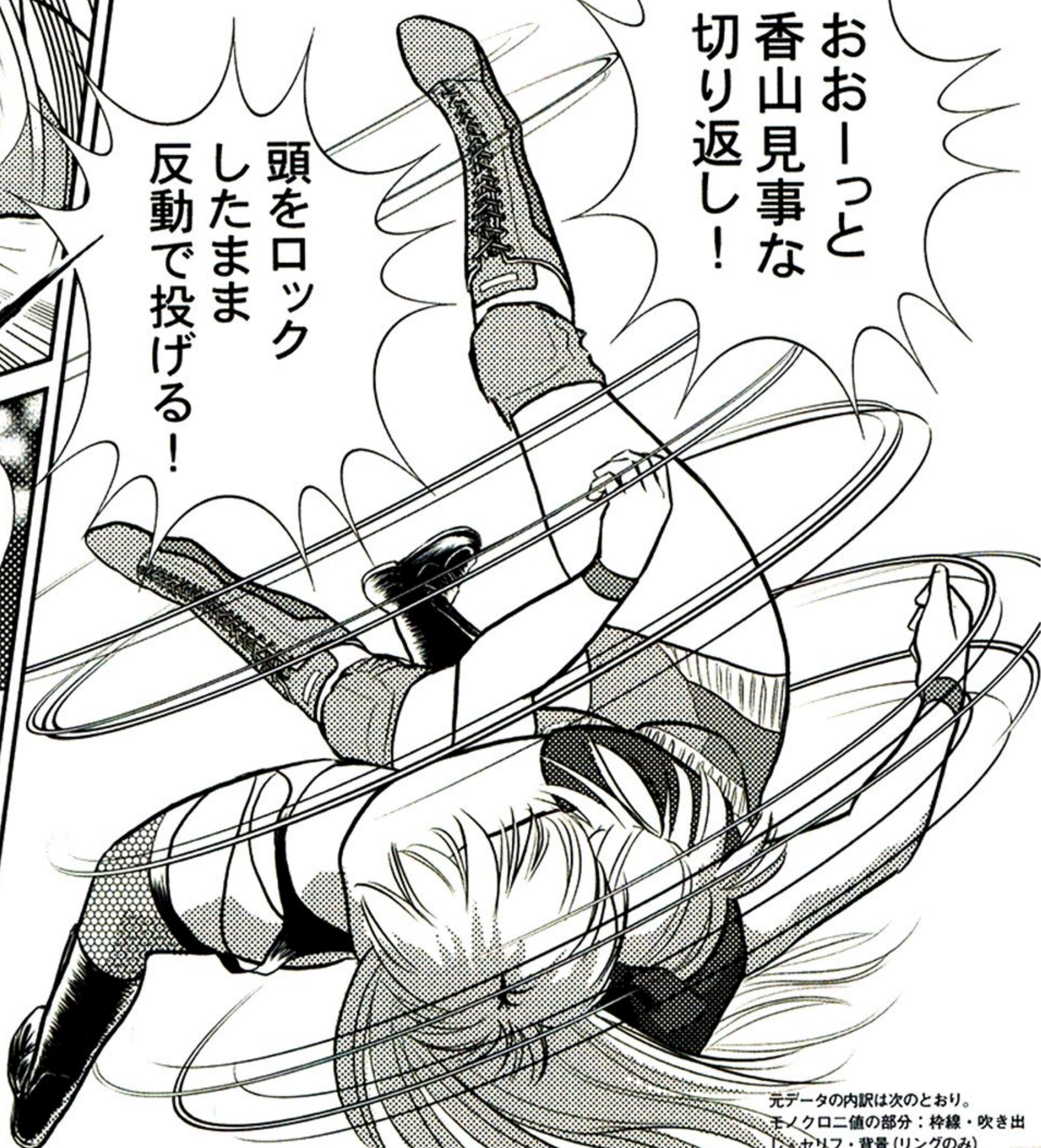
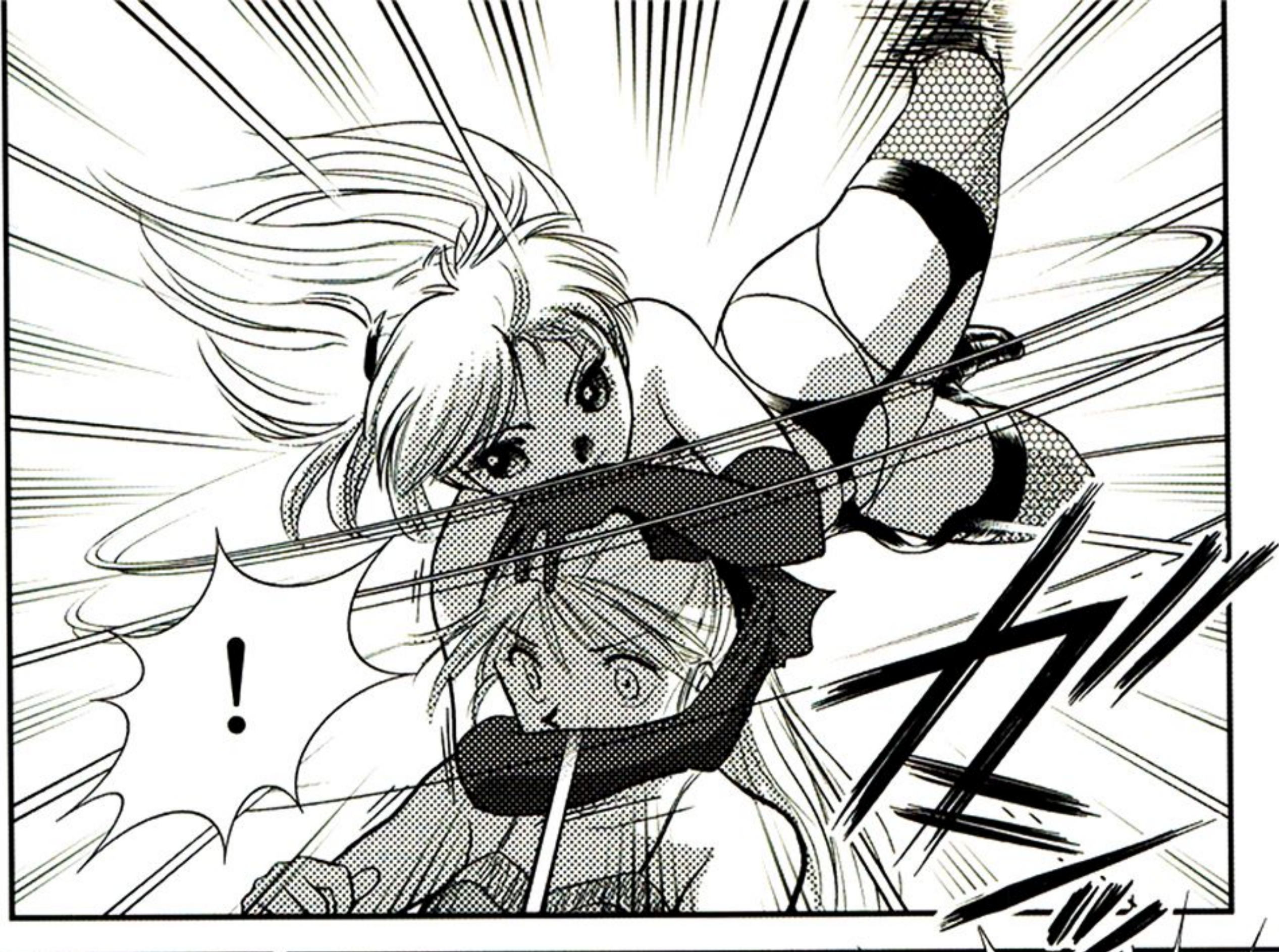
おおーっと香山見事な切り返し！

頭をロックしたまま反動で投げる！

鮎川油断したあっ！

ふふ…

香山すかさず足をロックしに行く！

B4に出力された原稿を90%縮小で掲載（次ページも同様）

こっ…これは！
足を極めたまま上体を反りアゴをロック
これはいわゆる鎌固めの状態だ
驚きました！サリー香山…見かけによらず技は本格的だ！
元プロレスラー黒岩征司の指導はダテじゃなかった！
ク……こいつ……
素人なんかじゃないわ！

元データの内訳は次のとおり。
モノクロ二値の部分：枠線・吹き出し・セリフ・背景（リングのみ）
アンチエイリアスのかかった部分：人物主線・スピード線・集中線・流線・アミトーン・タイツパターン・ウニ・書き文字・点描筆

600dpi相当のイメージデータをデータ入稿して印刷

はじめに断っておきますと，今回の原稿がOh!X誌面にどのように印刷されるかはあまり関知しません。私はいままでに一度もマンガ原稿をデータ入稿したことはなく，スクリーントーンのアミをどのくらいの解像度で出せば印刷にちゃんと出るかとかいったような知識はないからです。これからご紹介する作成手順は，あくまでも私が使用しているキヤノンのレーザープリンタLBP-750（メモリフル搭載）＋Mac用のドライバ「NetHawk」によってグレースケール600dpiの解像度で打ち出すことが前提となっています。いってみれば「生原稿」を作るのが目的です。デジタル入稿のことはわかりませんが，紙の原稿ならプロの現場で10年見てきてます。ただ，技法の一部はデータ入稿でも利用できるかと思います。

手元に使えるネームがなかったので，ネームは適当にでっち上げました。原稿は最終的にはIllustratorではなくPhotoshopから出力しています。扱うデータが大きいため，ある程度のCPUパワーとハードディスク上に1GB以上（できれば2GB）の作業領域が必要です。

原稿用紙の作成

市販のマンガ原稿用紙の寸法を手本にして，原稿用紙のテンプレートを作成します（図1）。

まずIllustrator上で書類をB4（幅257mm高さ364mm）に設定します。次に設定した書類にフィットするサイズの四角形を描き，それをメニューコマンドの「オブジェクト/トンボ/作成」でトンボにします。あとでPhotoshopデータで書き出すとき，このトンボで表される四角形（つまりいま描いた紙の大きさの四角形）のサイズでコンバートされます（データ入稿の際にはこのトンボを含めた大きさが必要だそうですが）。そして市販の原稿用紙を参考にしてどこまで印刷されるかの目安となる四角形を描き，今度は「フィルタ/クリエイト/トリムマーク」でトンボをつけます。これはトンボを実際のオブジェクトとして描画するもので，Photoshopデータにコンバートしたとき，どこまで絵を描くかの目安にするだけで，レイヤーごと不可視にしてプリントはしませんから，別にトンボでなくても四角形などで構いませんが，そのうちトンボを打ち出す必要も出てくるかと思ってこのようにしました。

さらに内側の270mm×180mmの枠に沿った線を入れます。私の場合はなにかの役に立つかもしれないと思って内側の四角形を3等分あるいは4等分する線を入れてあります。これらの線は「画面/ガイドを作成」ですべてひとつのレイヤーに収めてガイド専用のレイヤーとしてレイヤーごとロックします（ちなみにレイヤー化しないでも「画面/ガイドをロック」でロックすることはできる）。

次にコマを割るための基本となるオブジェクトを新規レイヤーに用意します。スマートガイド機能を利用すれば作成したガイドに対して簡単にオブジェクトをスナップさせることができます。まず基本枠の270mm×180mmの四角を作成します（図2）。これがコマを割るための基本の四角です。線の太さは0.6mmにしています。原稿用紙をロードしてすぐコマ割り

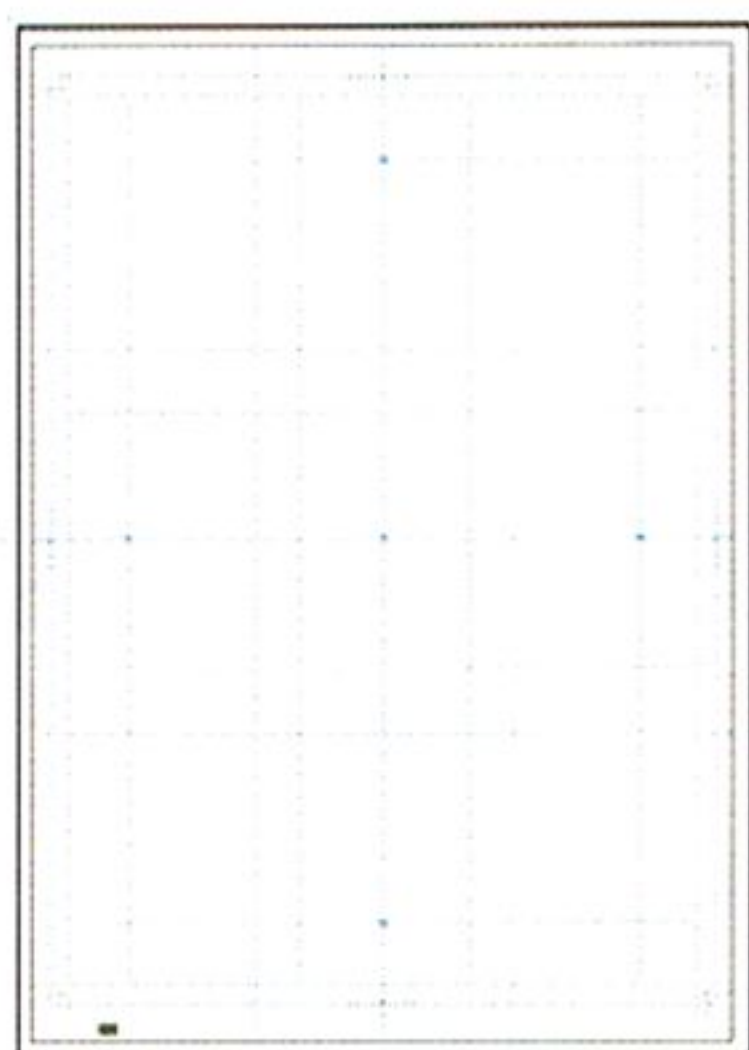



図1　市販のマンガ原稿用紙をもとに原稿用紙のテンプレートを作成。外側のトンボはPhotoshopデータに変換したときにそのトンボで切られたサイズの書類になる。内側のトンボは「フィルタ/クリエイト/トリムマーク」で作成。データ入稿の場合，外側のトンボも「トリムマーク」で作成する

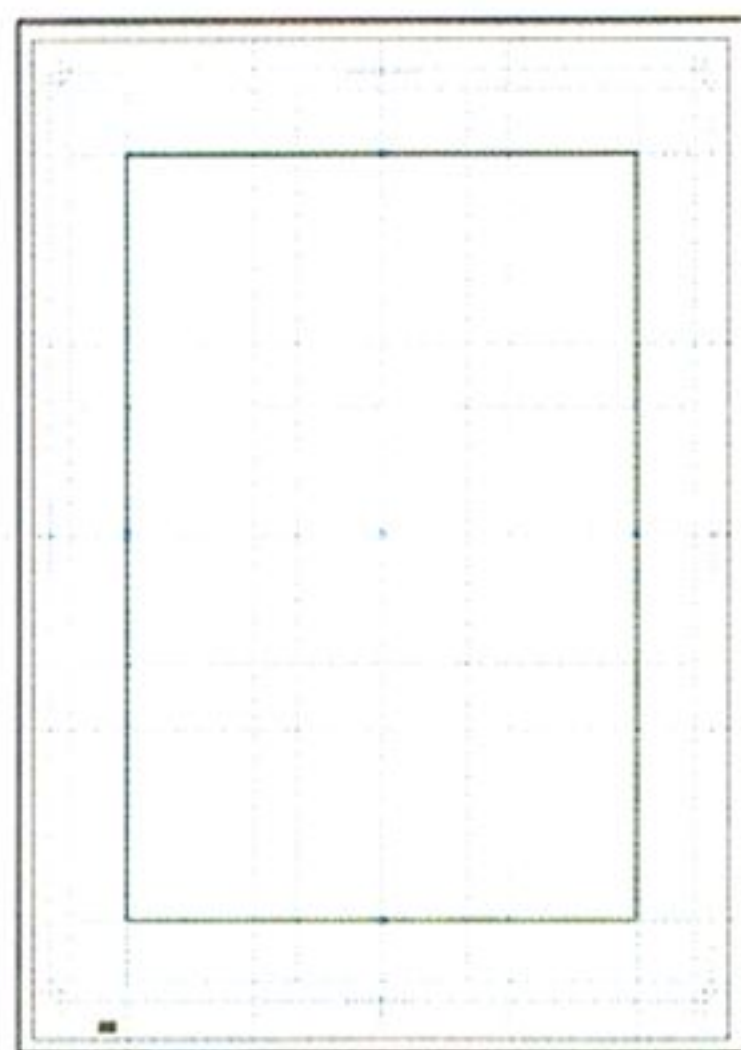



図2　コマ割りの基本枠を描く

図3　コマ割り棒を配置しておく。この状態で保存して「ひな型」ファイルにする

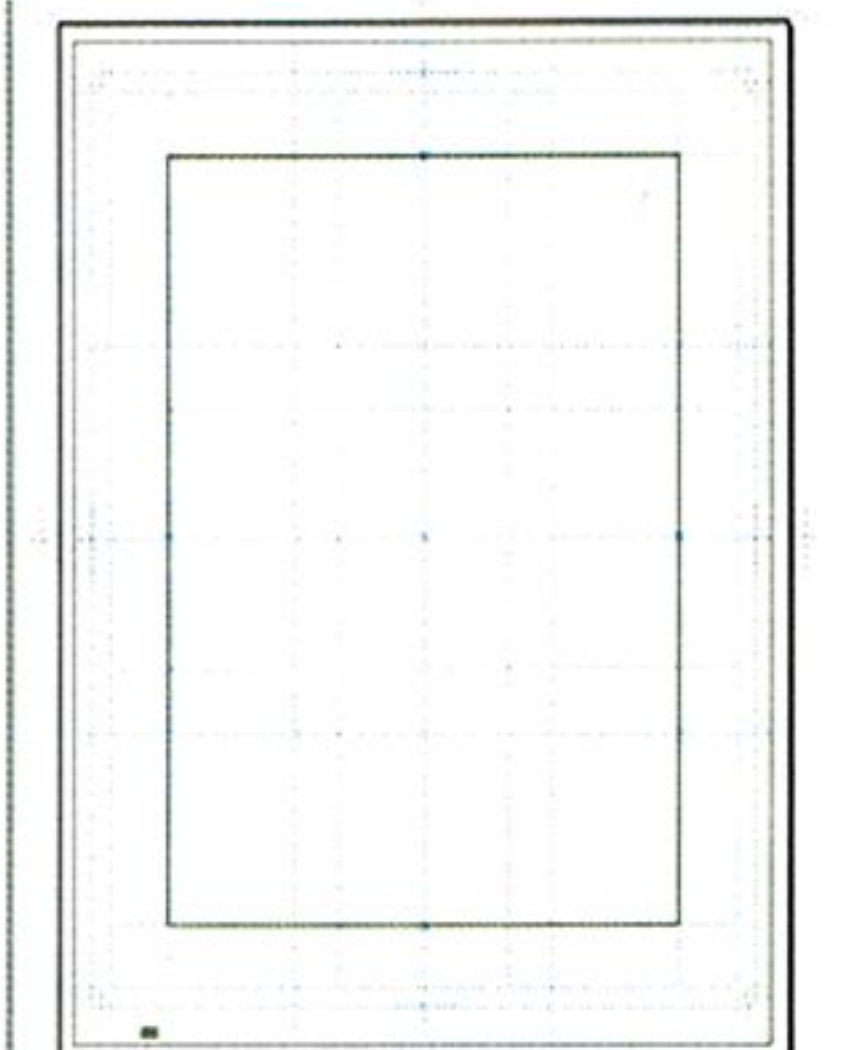



図4　下絵を配置。レイヤーをロックしておく

図5　下絵にあわせてコマ割り棒をコピーして基本枠の上に配置

図6　前面オブジェクトで型抜き。基本枠をコマ割り棒で型抜きする

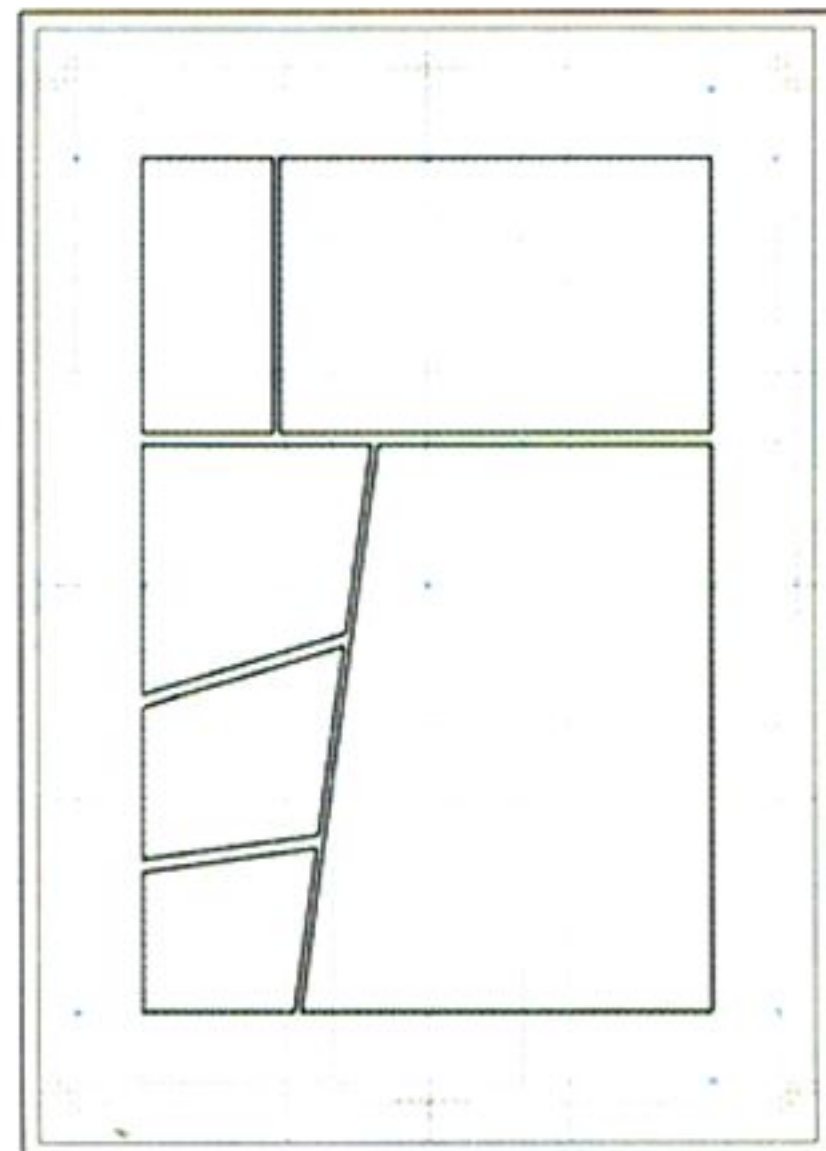



図7　下絵を隠した状態。コマが割られている

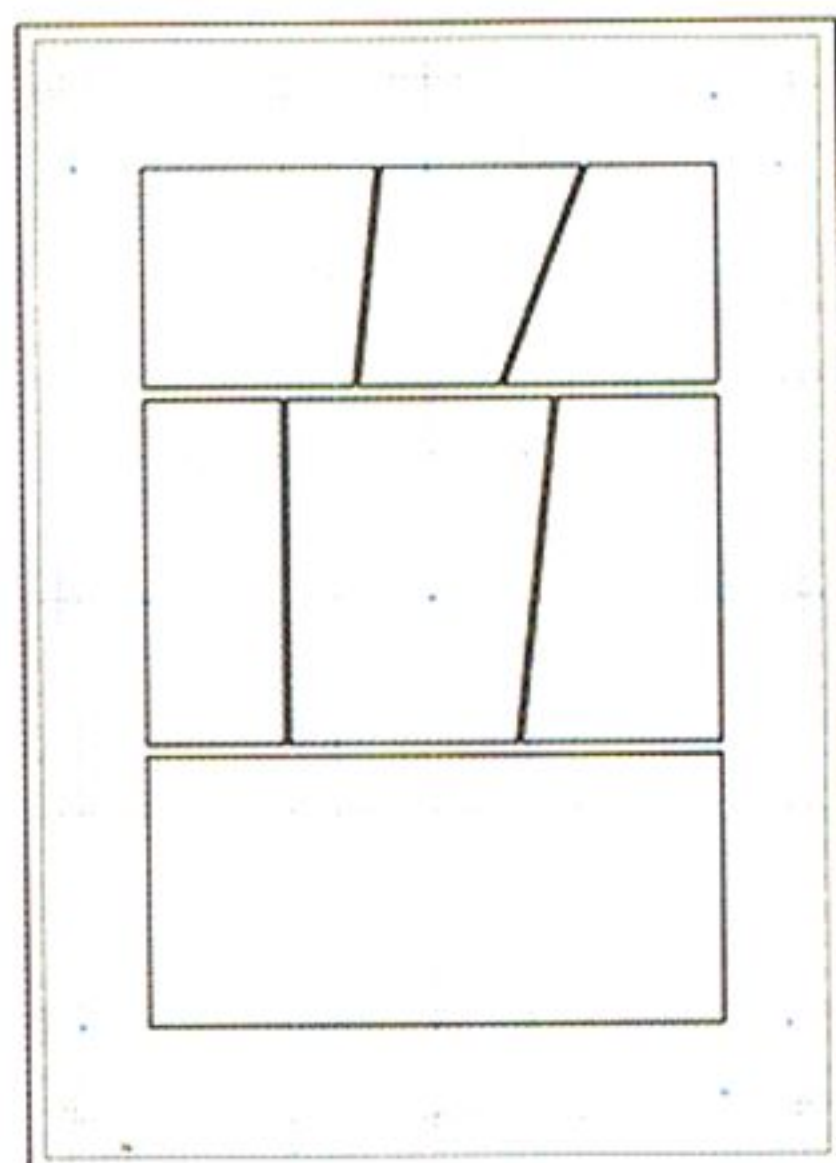



図8　上段の真ん中のコマを……

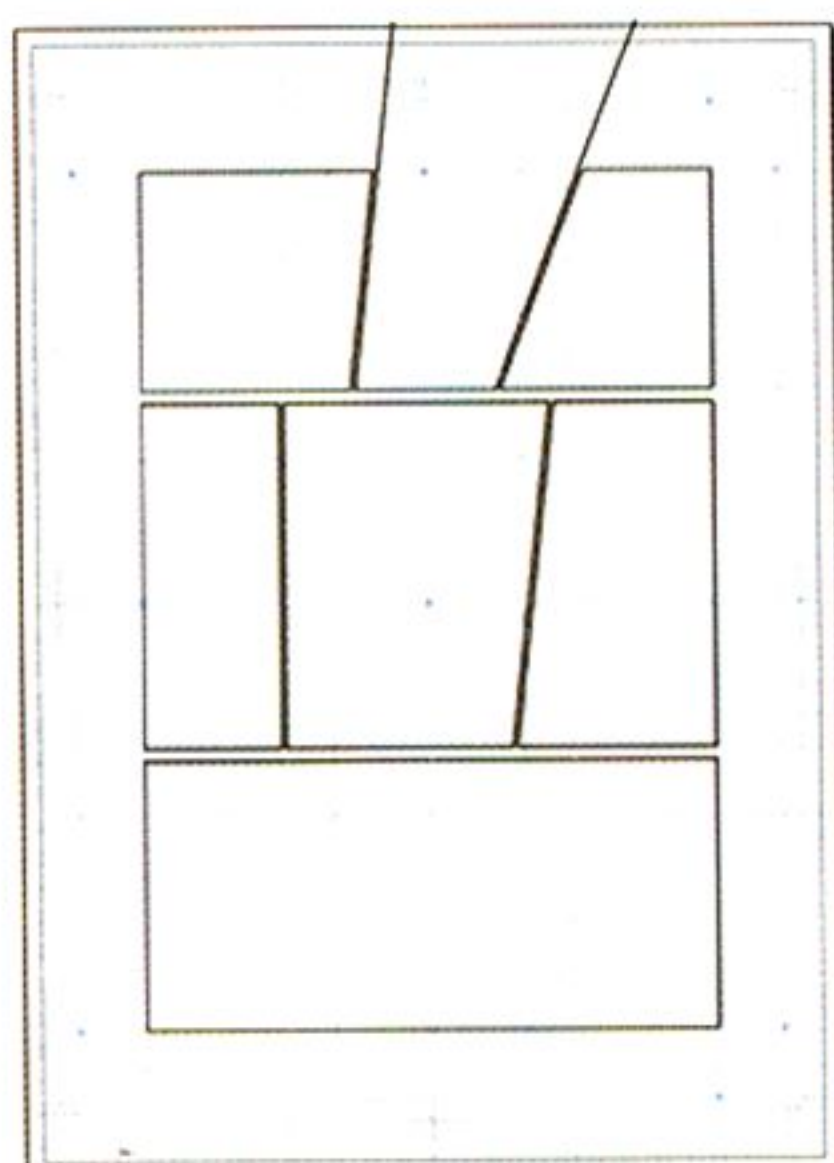



図9　このように延長するにはコツが必要

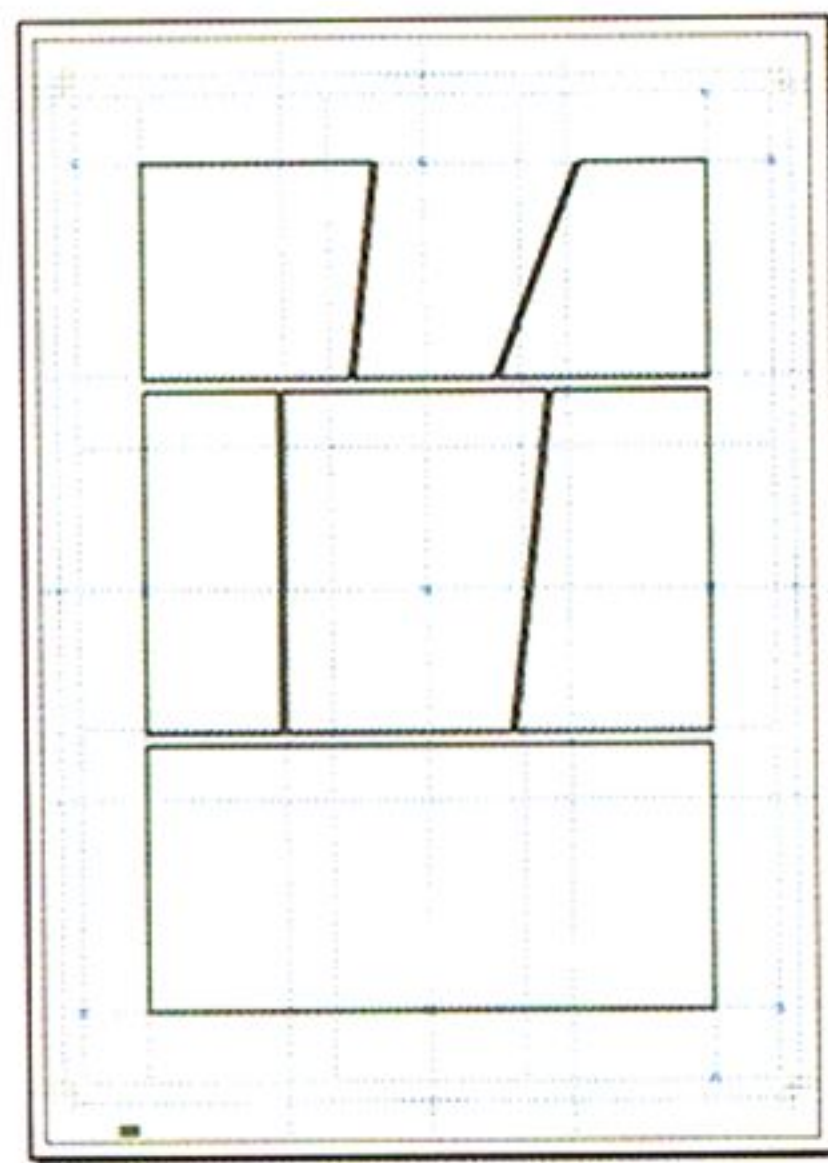



図10　いらない線を削除。説明のために余計な線を削除しただけで，本当は削除しなくてもよい

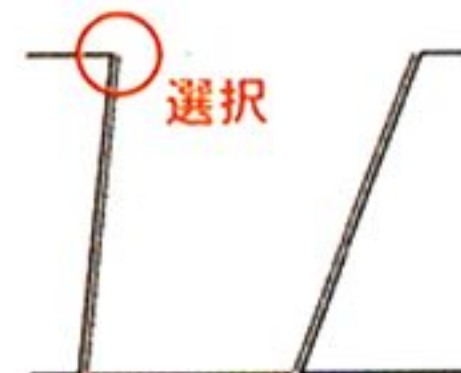



図11　延長したい線の端を選択

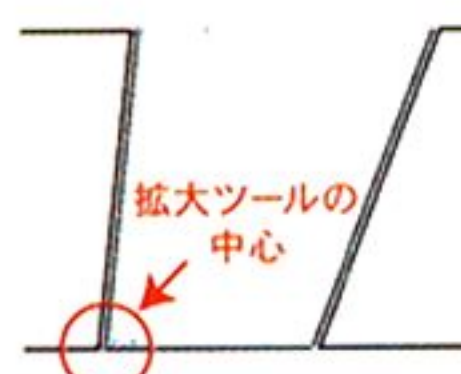



図12　拡大ツールを選択して最初のクリックで延長したい線の反対側をクリック。拡大の中心を決める

図13　Shiftキーを押しながら図のようにドラッグ。スマートガイドがオンになっていればShiftを押さなくても「縦横比を固定」という表示が出るようにドラッグすればよい

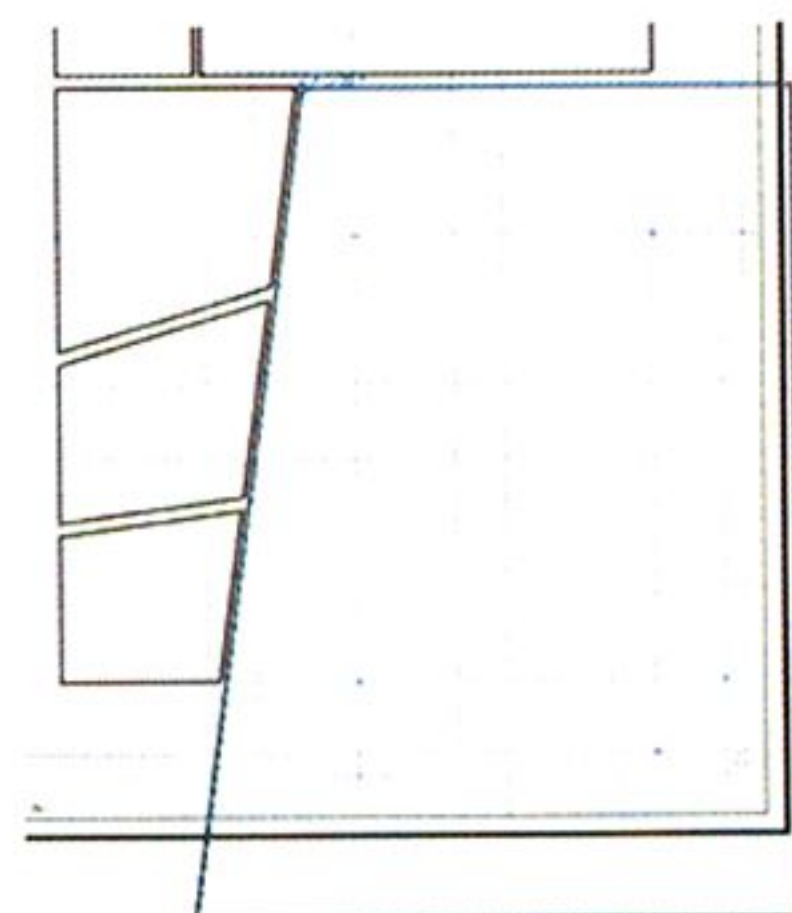



図14-1　左上隅を中心点として拡大。「線幅も拡大・縮小」しないように注意

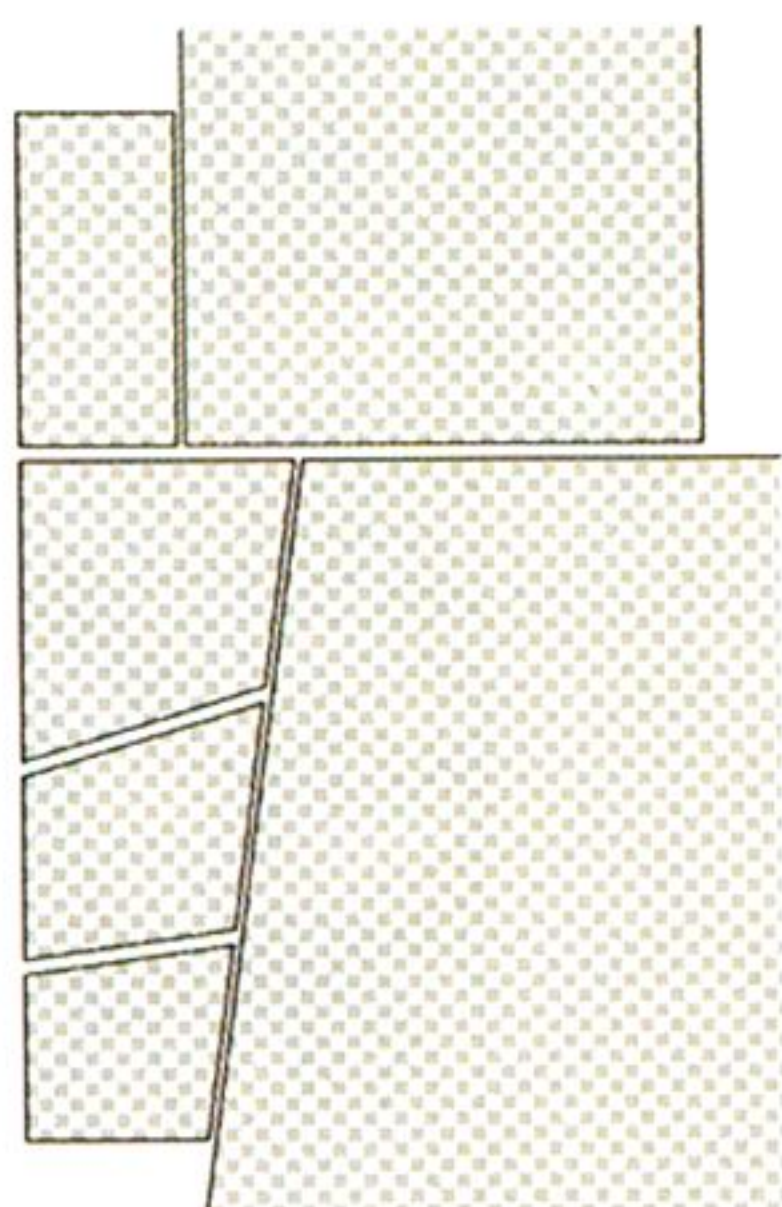



図14-2　Photoshopデータにコンバートした状態。コマの中が透明になっている

図15　多角形を描く

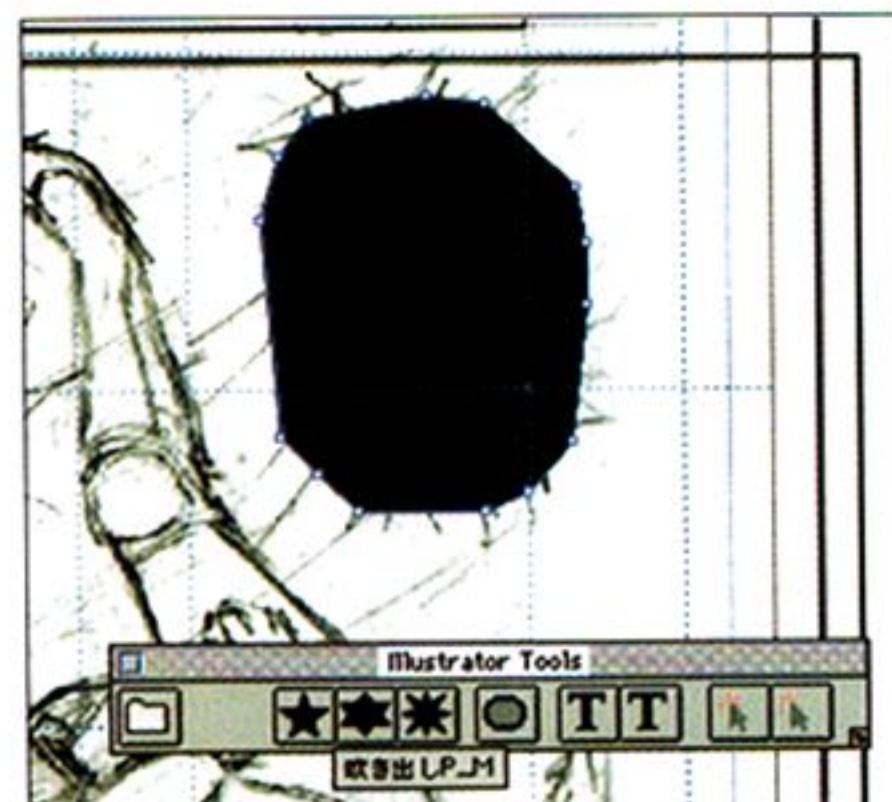



図16　Quickeysで作ったツールバーのボタンを押す

に取り掛かれるようにあらかじめ線の太さが同じく0.6mmの細長い棒状の四角を左側と下側に用意してみました（図3）。縦の棒は幅が1.5mm～2mm，横の棒は幅（縦の長さ）4mmです。線の太さや分割棒の幅は人それぞれです。これを使ってコマを縦横に割っていきます。

この状態で「マンガ原稿用紙」と名前をつけて保存し，Finderの情報を見るコマンドでひな型ファイルにチェックを入れておきます。これで次にダブルクリックで開いたときは新たなファイル名で開かれ，元のファイルが失われることはありません。

コマ割り

まずスキャナで取り込んだ下絵をPhotoshopデータで保存し，Illustrator上で新たなレイヤーに読み込んで配置し，レイヤーをロックしておきます（図4）。次にあらかじめ作成して左側と下側に配置しておいた縦横のコマ分割線をコピーして（オプションキーを押しながらドラッグ）270mm×180mmの基本枠の上に配置します（図5）。斜めに傾いている線はダイレクト選択ツール（白い矢印の選択ツール）を駆使して隣のコマにはみ出さないようにします。ただし，基本枠の外側にはいくらはみ出しても構いません。

そうしておいて全オブジェクトを選択し，パスファインダの「前面オブジェクトで型抜き」を実行します（図6）。新たに作成したオブジェクトは常に前に作成したオブジェクトの上に置かれますから，原稿用紙のテンプレート作成の段階で基本枠が分割棒よりも背面（最背面）になくてはなりません。下絵を隠すと図7のようになります。

断ち切りのコマがある場合はダイレクト選択ツールで引き伸ばす部分を選択してShiftキーを押しながら一方向に引き伸ばせばよいのですが，たとえば，図8のように斜めになったコマを図9のように延長するには少しコツがいります。まずダイレクト選択ツールでいらない線を選択して削除します（図10本当は削除する必要はありませんが）。次に同じくダイレクト選択ツールで延長したい線の端の点を選択します（図11）。その状態で拡大ツールを選択し，最初にそのコマの延長したい線のもう一方の端をクリックします（図12）。そこが拡大の中心点となります。スマートガイドが有効になっていればスナップさせる点に「アンカー」という文字が出るはずです（「ファイル/環境設定/スマートガイド」の「ガイドのヒント表示」がチェックされている必要があります）。中心点をクリックしたらもう一度どこかほかの点をクリックして，その点と拡大の中心点を結ぶ延長線方向にShiftキーを押しながらドラッグして拡大を実行すれば，線が角度を保ったまま延長されます（図13）。

このようにすれば断ち切りゴマも簡単に作れます。実はスマートガイドのヒント「縦横比を固定」に沿ってドラッグすればShiftキーを押す必要さえありません。今回の作品の場合は「前面オブジェクトの型抜き」を実行したあと，グループを解除して右下のコマを選択しそのコマの左上隅を中心点にして拡大ツールを使用しました（図14）。この場合は　拡大ツールのオプション「線幅も拡大・縮小」がoffになっていなければなりません。

これでコマ割りが終わったわけではありません。さらに線幅0.6mmで塗りは白，今度は原稿用紙の外側にはみ出すサイズの四角形を上に描きます。そして全オブジェクトを選択し，パスファインダ「背面オブジェクトで型抜

図17 終了メッセージ

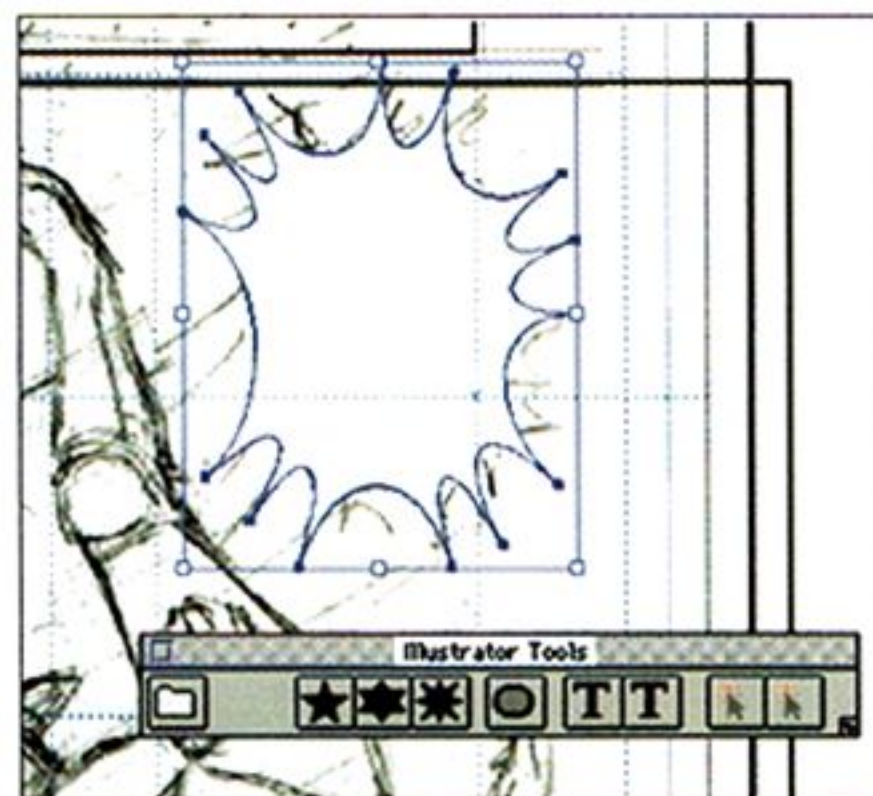



図18 パンク吹き出しができている

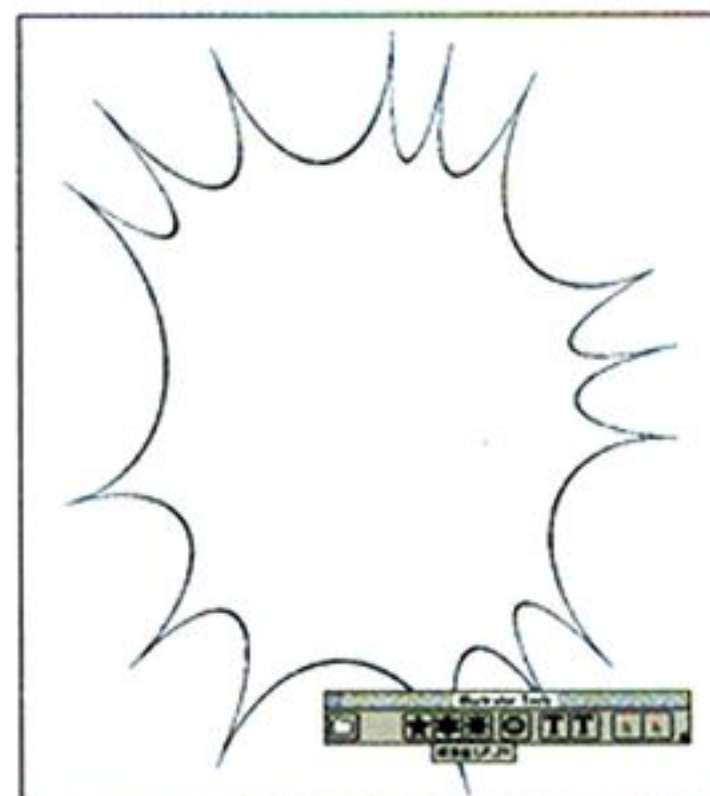



図19 筆圧ペンで描いたようなタッチになっている

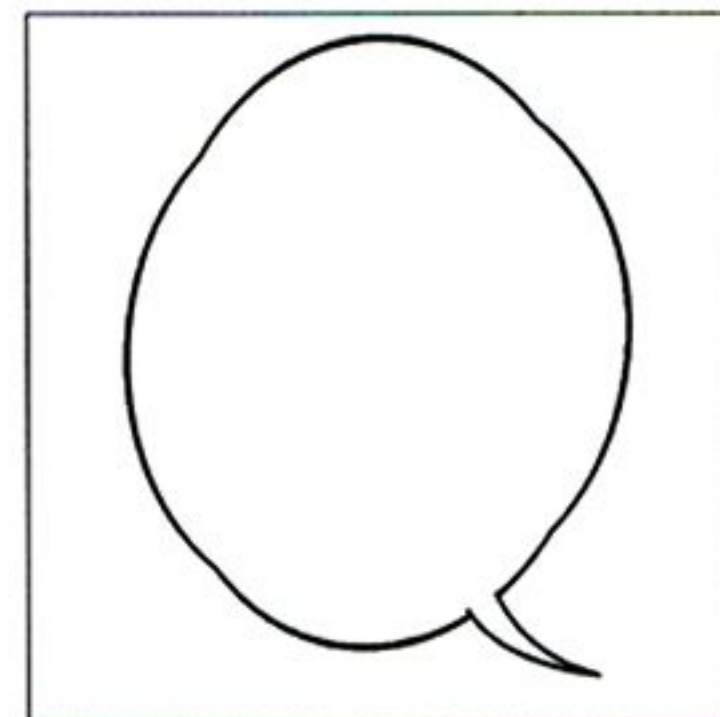



図20 同じような理屈でバルーン吹き出しもできる

図21 セリフを入れる。昔，出版社に持ち込みする前に書き文字を清書したら，それだけで一日潰れました

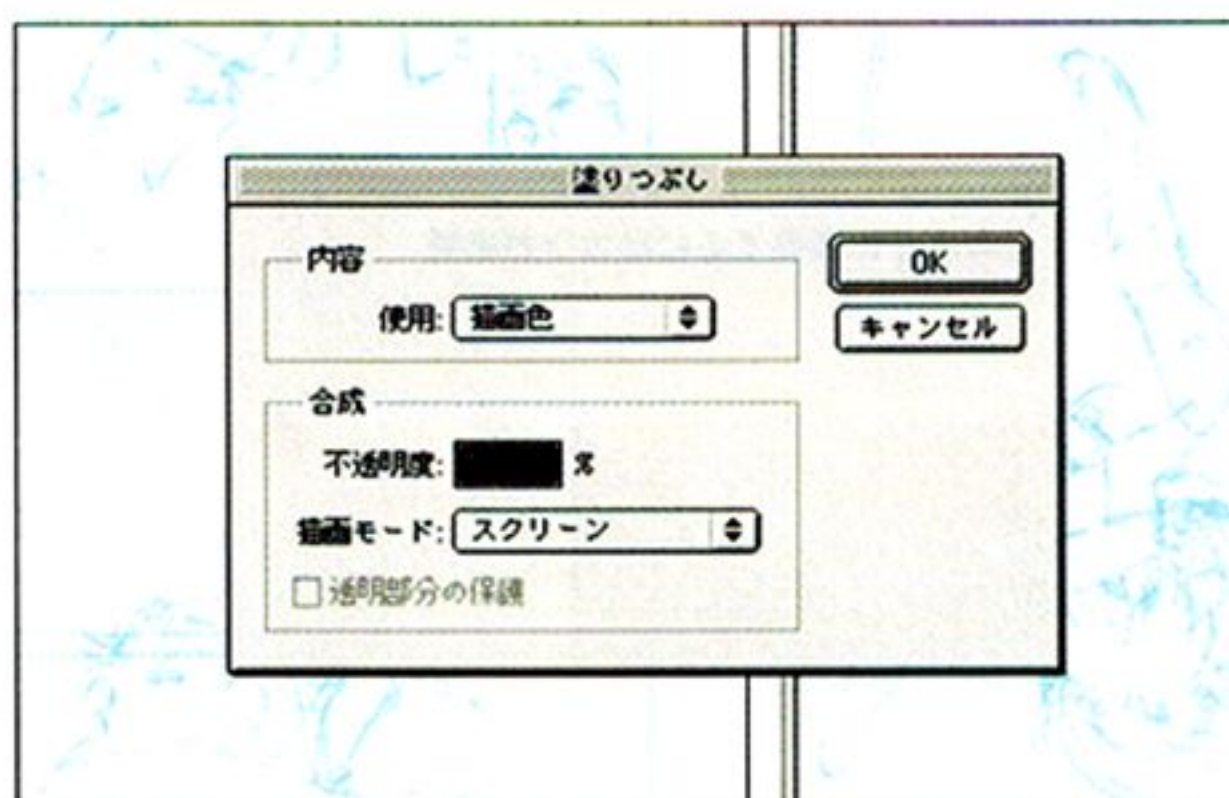



図22 下絵の線を薄い青に変える。枠線も下絵と合成して薄青にしておくと，主線を入れるときに目安となる

図23 Painter上でペン入れ

き」を実行します。

結局のところ線と塗りの属性は最後に描いた四角形のものが継承されますから，実をいうとそれ以前のコマ割りオブジェクトには線や塗りの属性は必要ないのです。最終的にPhotoshopデータにコンバートした枠線は図15のようにコマの中が透明になります。これでようやくコマ割りの完成です。文章で書くと長いですがほんの数分で終わる作業です。

吹き出し

私のIllustratorには世界にひとつしかない「塗りと線が黒で線の太さが0.2～0.25ポイントのいい加減な多角形」を描いてボタンひとつ押せば綺麗なパンク吹き出しに加工してくれるというありがたい機能がついています。実はこれはQuickeysというコンピュータ上の定型作業を自動化してくれるツールによって実現しています。本来は定型作業をショートカットキーに割り当てるソフトウェアですが，バージョン3.5からアプリケーションに連動して起動することができる「ツールバー」に定型作業を割り当てられるようになり，Illustratorにあたかも最初から備わっている機能であるかのような感覚で使えるので非常に重宝します。実際の使用法は次のようなものです。

①塗りの色を「黒」，線の色を「黒」太さを0.2～0.25ptにする
②その属性で適当な多角形を描く（図15）。コツとしてはトゲになる部分にポイントを二三個ずつ集中しておくとパンク吹き出しっぽくなる
③ツールバーのボタンを押す（図16）
④終了のメッセージが出たら（このダイアログはQuickeysで作ったもの）リターンキーを押す（図17）
⑤塗りが白で線が黒のパンク吹き出しができている（図18）

ボタンを押すとほんの数秒でできあがりです。Illustratorには標準でパンクフィルタがついていますが，私のパンク吹き出しのウリはペンで描いたようなタッチがついていることです（図19）。実はこれは微妙にパンクの度合を変えた白と黒のオブジェクトを2枚重ねているのです。Quickeysにおける実際の作業手順の内訳を次に示します。なお，この手順は手作業でやると手描きでパンク吹き出しを描くより時間がかかります。

①目的のオブジェクトを選択し，塗りを黒，線を黒，線の太さを0.2～0.25ptにする。（前準備。Quickeysではこの作業は困難。②からがQuickeysの作業）
②「編集/選択/未選択オブジェクト」を実行（選択の反転）
③「オブジェクト/ロック」（目的のオブジェクト以外をロック。作業を自動化する際，2つになったオブジェクトの選択を切り替えるにはほかのオブジェクトが邪魔）
④「編集/選択/未選択オブジェクト」を実行（選択の反転。目的のオブジェクトを選択状態に）
⑤「フィルタ/パンク・膨張」値は「-28」
⑥オブジェクトをコピー
⑦「編集/前面へペースト」を実行（ペーストした前面のオブジェクトが選択状態）
⑧「オブジェクト/変形/拡大・縮小」値99%（100%である下のオブジェクトとの差が吹き出しの線の太さになる。99%ではちょっと細いかも）
⑨（本来ならここで「フィルタ/パンク・膨張」を縮小したオブジェクトに対して実行し，下のオブジェクトと度合を変えたパンクをかけるのだが，この場合，縮小した時点でパンクのかかり具合が微妙に変わっているので必要がなくなった）
⑩塗りを白に（上のオブジェクト）
⑪全オブジェクトを選択（ほかのはロックされているので実際には重なった2つのオブジェクトが選択される）
⑫グループ化
⑬「編集/選択/未選択オブジェクト」（選択の反転）

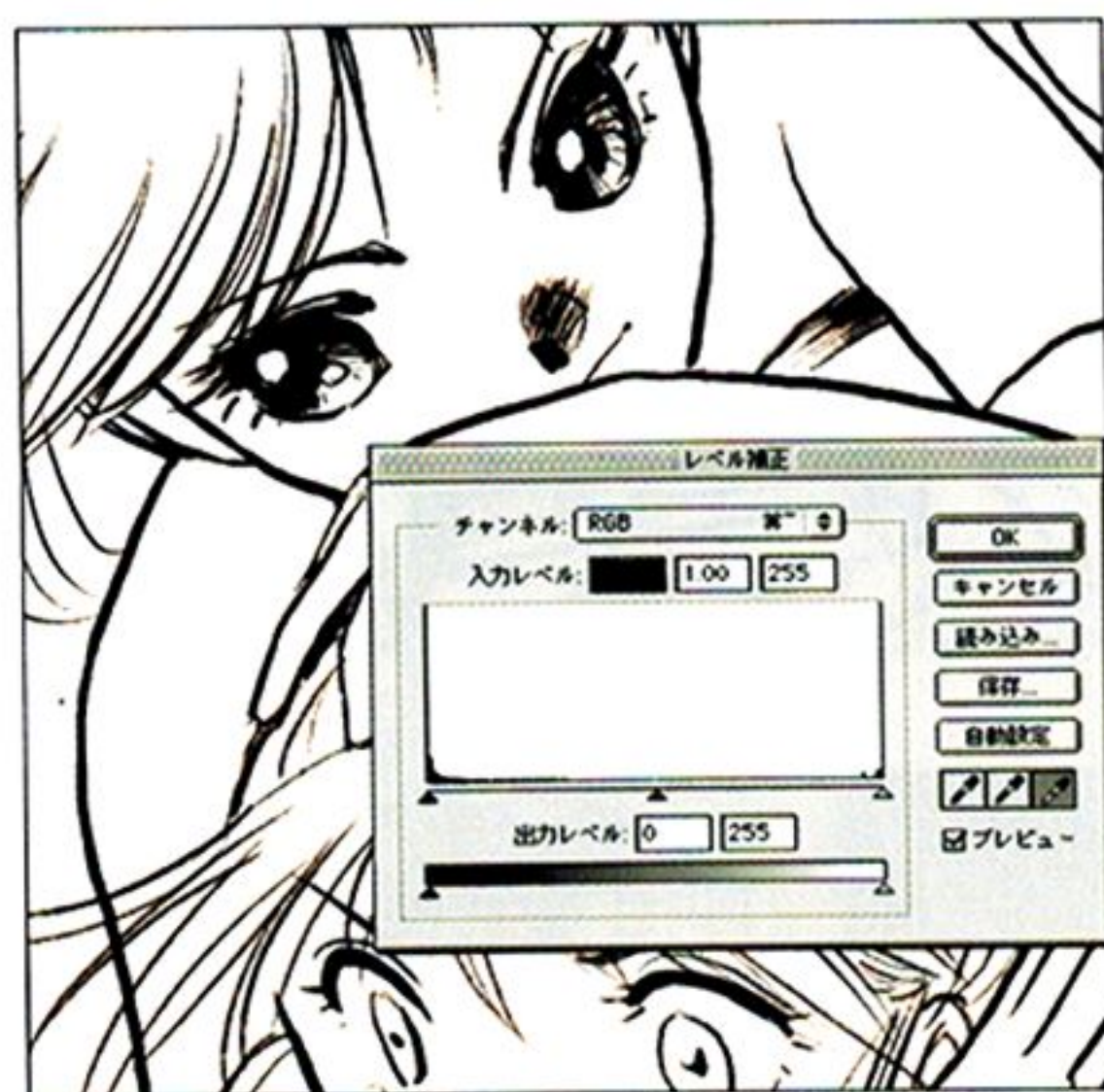



図24　下絵の青を飛ばす

図25　書き文字は別レイヤーに書いたもの。選択範囲を広げて縁を白で塗る

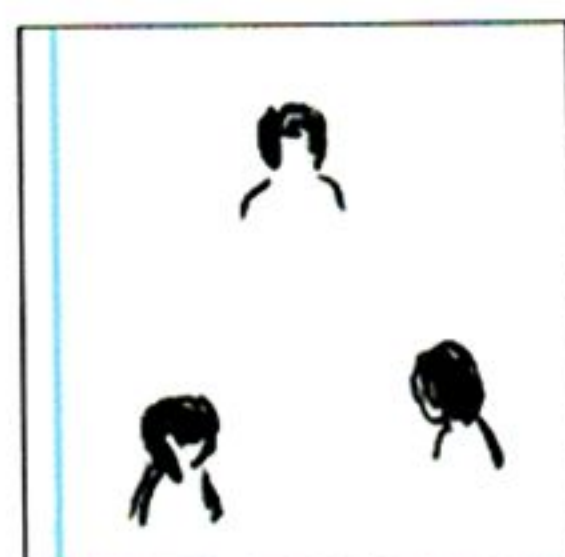
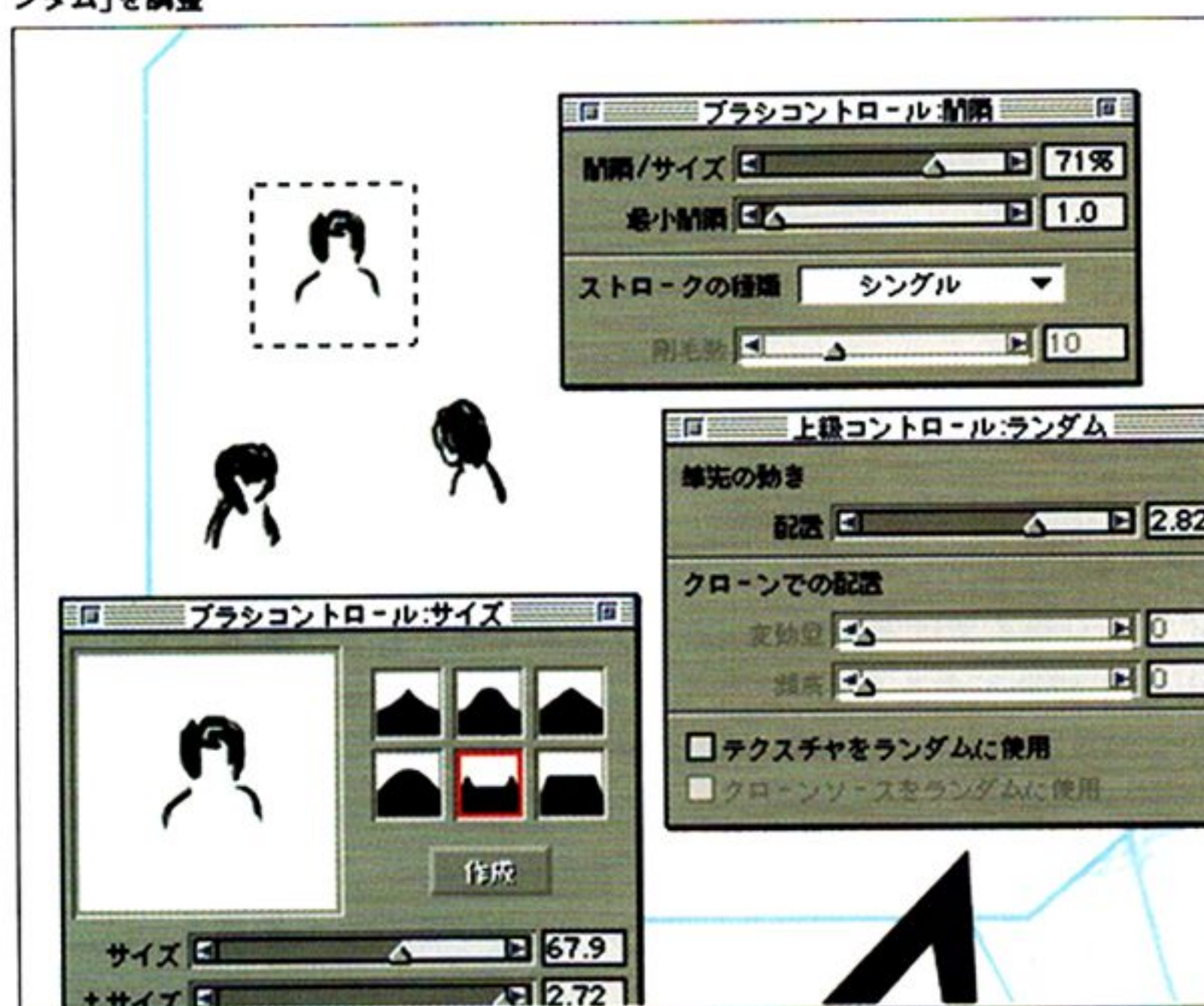



図26　筆として取り込むための基本形状を描く。正面向き・右向き・左向きの3種類

図27 「ブラシの取り込み」を実行し，ブラシコントロールの「間隔」「ランダム」を調整

⑭「すべてをロック解除」
⑮「編集/選択/未選択オブジェクト」(できあがったオブジェクトを選択状態にする)

以上です。縮小率を変えることによって吹き出しの太さを，パンクの数値を変えることでペンタッチの具合を調整できます。なお，実際にQuickeysでシーケンスを作成する際には「処理のダイアログウィンドウが出てから数値を入力し，数値の入力が終わってからOKボタンを押す」といった順番を確実に実行させるため，間に適当なウエイトをおく必要があります。それと，最終的にすべてのオブジェクトのロックを解除してしまいますから，それ以前にロックしていたオブジェクトがあってもロックが解除されてしまいます。これらと関係なくロックしておきたいオブジェクトは別レイヤーに用意して，レイヤーごとロックしておく必要があります。

ちなみに通常のバルーン吹き出しも同じような方法で作成していますが，こちらのほうはしっぽをあとからペンで継ぎ足したりなどして，あまりカッコのいいものではありませんので，ここでの紹介は控えます(図20)。

ところで，Illustratorもバージョン8.0からPhotoshop同様アクション機能が使えるようになりました。このパンク吹き出し機能を作ったときはまだ，Illustratorにはアクション機能がありませんでしたのでこの原稿を書いたのをきっかけにアクション機能に移植してみましたが，さすがに専用ツールだけあって，Quickeysではタイミングをとるのに処理の間に適当なウエイトをおく必要があったため5～6秒かかっていた作業が，アクション機能では一瞬で終わってしまいました。しかも塗りや線の属性まで気にしなくてよくなったため(前準備がなにもいらない)，格段に作業が楽になりました。

セリフを入れる

テキストツールを使用してセリフを入力します(図21)。これを手書きで書く手間もなかなかバカにはなりません。上から写植を貼るのであればフォントや文字詰めに気を遣う必要はないでしょうが，同人誌など，ここで入力した文字がそのまま出力に使用される場合はコマンド＋シフト＋［(］などを駆使してある程度は文字詰めにも気を遣ったほうがいいでしょう。

枠線の書き出し

枠線・吹き出し・セリフはPhotoshop上で扱いやすいようにそれぞれ別レイヤーになっています。これをPhotoshopデータに書き出すときにアンチエイリアスをかける設定にしてあると時間がかかりすぎて実用的ではありません。600dpiなら必要十分のクオリティが得られますからアンチエイリアスのチェックは外しておくことをおすすめします。これならレイヤーを保持したままでも十分な速度で書き出せます。

人物のペン入れ

書き出した枠線と取り込んだ下絵をPhotoshop上で合成して，必要があれば下絵のキャラクターなどを再配置してレイアウトを整え，画像を統合します。そして下絵の線を「塗潰し/描画色/スクリーン」で薄い青に変えます(図22)。それを別名で保存し，今度はPainter上に持っていって，前回作成したペンで人物にペン入れをします(図23)。もちろんPainterでペン入れするのは慣れが必要ですから，通常はペンで描いたものをスキャナで取り込むのがいちばんですが，ここでは可能性を模索する意味で，Painterを使用しています。書き文字などは手っ取り早く透明レイヤー上に透明レイヤー専用の筆やペンを使って描いています。

ところで600dpiでB4のサイズだとグレイスケールなら問題ないのですが，Painter上ではカラーとして扱われるため，レイヤーを使ってなくてもデータが大きすぎて処理が重く，とても扱いきれません。そのため400dpiに落としたつもりなのですが，それでも処理が重かったので結局下絵は上下2つに分割してペンを入れてます。

ペンが入ったら例によってPhotoshopのレベル補正で下絵の青を飛ばし(図24)，レイヤーを統合しない設定でいったんグレイスケールに変換して色を落とします。書き文字は別レイヤーになっていますから，文字部分を選択して選択範囲を拡張し，「塗潰し/白/乗算」で縁を白く塗ります(図25)。

リング周辺のちっちゃなモブはPainterの「ブラシの取り込み」機能を使用してモブ用の筆を一時的に作りました。最初に基本となる人形を大雑把に描いたら(図26)長方形選択ツールで選択してブラシウィンドウのウィンドウメニュー「ブラシ/ブラシの取り込み」を実行します(図27)。ブラシのバリアント設定はそのときアクティブになっているブラシの設定が引き継がれますから，この場合はスクラッチボードなどよく使用する普通のペンを選択しておきます。人形をまばらに配置するためには「ブラシコントロー

図28 ブラシを左から右に走らせると一筆でこのようになる。多少筆圧をサイズに反映させている

図30 リングのパースとあわせるためのスクリーンショットを取る

図31 Shade上で透視図にスクリーンショットを読み込み，透視図のみの表示にしてパースをあわせる

図29 正面向き・右向き・左向きそれぞれを筆として取り込んで，列をなぞるように描く

図33 アクション機能を利用して，ひとつの画像を加工した4種類の画像を生成

図34 元画像をコピーして「エッジのポスタリゼーション」をかけたもの

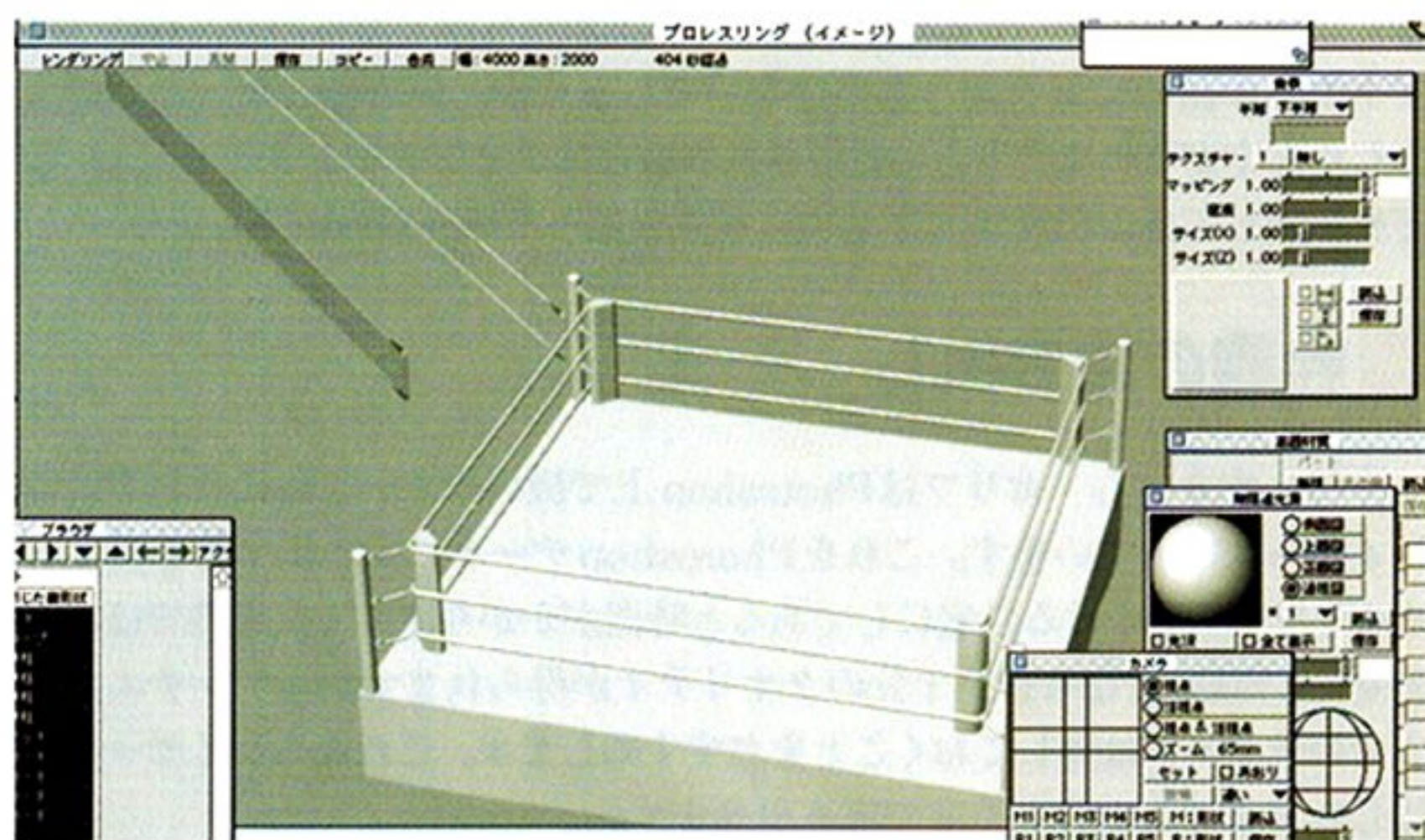



図32 エッジを拾いやすい画像になるように設定してレンダリング。背景はグレイにしてある

図35 いちばん上の画像。fig035.tiffの画像を二値化したもの。もっともしきい値が高い

ル：間隔」の「間隔/サイズ」の値を大きく取って，ランダム「上級コントロール：ランダム」の「筆先の動き/配置」の値を適当に上げておきます。この設定でさっと一筆走らせるだけで「黒山の人だかり」を簡単に描くことができます（図28）。このようにしてできたのが図29です。

できあがったものは再び600dpiに変換して原稿に配置します（「画像の再サンプル」のチェックを忘れないように）。最初から600dpiで画像を分割して描くか，400dpiよりもっと解像度を落として1枚絵で描くか，それはもちろん描く人の自由です。

背景の作成

さらに試みとしてプロレスのリングを前回同様ShadeProfessionalR3で作成しています。というより3Dを背景に利用する技法はもともとマンガに利用することが目的でした。3Dで背景を作っておけば同じ背景をいろんな角度から描くことができるというメリットがあります。しかしながら，立体のオブジェクトを作るのはそれなりに時間がかかりますし，同じ背景というのは意外と出てこないものなので，すべての背景に利用するというのはやはり無理があります（あくまでも個人レベルでの話で，組織的にやれば十分意味はあるはず）。しかし，このようにプロレスがメインのマンガだったりするとリングは繰り返し出てきますから，それなりに利用価値はあります。

Shade上でパースをあわせるための下絵としてリングのあるコマのスクリーンショットを取ります（図30）。モデリング自体は単純な基本形状の集まりなのでそんなに時間はかかりません（ディティールはあとで描き加えればいいので）。リングができたらパースペクティブ画面に下絵を読み込みパースをあわせます（図31）。レンダリング設定は「スキャンライン」で「材質を表示＝on」「背景を表示＝on」「背景を反映＝off」「曲面の分割＝細かい」「アンチエイリアス＝on」「影を表示＝off（スキャンラインでは常にoff）」で，イメージサイズは2000×4000ピクセルです。背景が黒ではあとでPhotoshop上で加工するとき，アウトラインのエッジを拾えませんから別の色にしておきます。レンダリング画像は図32のようになります。

レンダリングした画像をPhotoshop上で加工する手順は前回と全く同じです。私は前回の画像加工過程をPhotoshopのアクション機能に記録しています。ボタンひとつでひとつのレンダリング画像から処理をかけた画像を4枚生成しますが，その内訳は図33のようなものです。元画像を複製したレイヤーに対し，エッジのポスタリゼーションをかけたもの（図34）とそれを異なるしきい値で二値化した3種類の画像。この二値化した3枚から使える画像を選ぶのですが，たいていはしきい値の高いほうから1枚目（図35）か2枚目（図36）の画像を選ぶことになりますが，3枚目（図37）の黒い画像もときとしてオブジェクトのマスクと併用することでオブジェクトにスクリーントーンを貼るための選択範囲として利用できます（利用例図38）。

実際に使用したのは2枚目の画像です。このレイヤーを原稿の方に放り込むわけですが，その前にレンダリングのときについてきたマスクをアルファチャンネルから選択範囲として読み込み（図39）これをレイヤーマスクとして加えます（図40）。

ドラッグ＆ドロップでリングのレイヤーを原稿の方に放り込んだら，すぐ

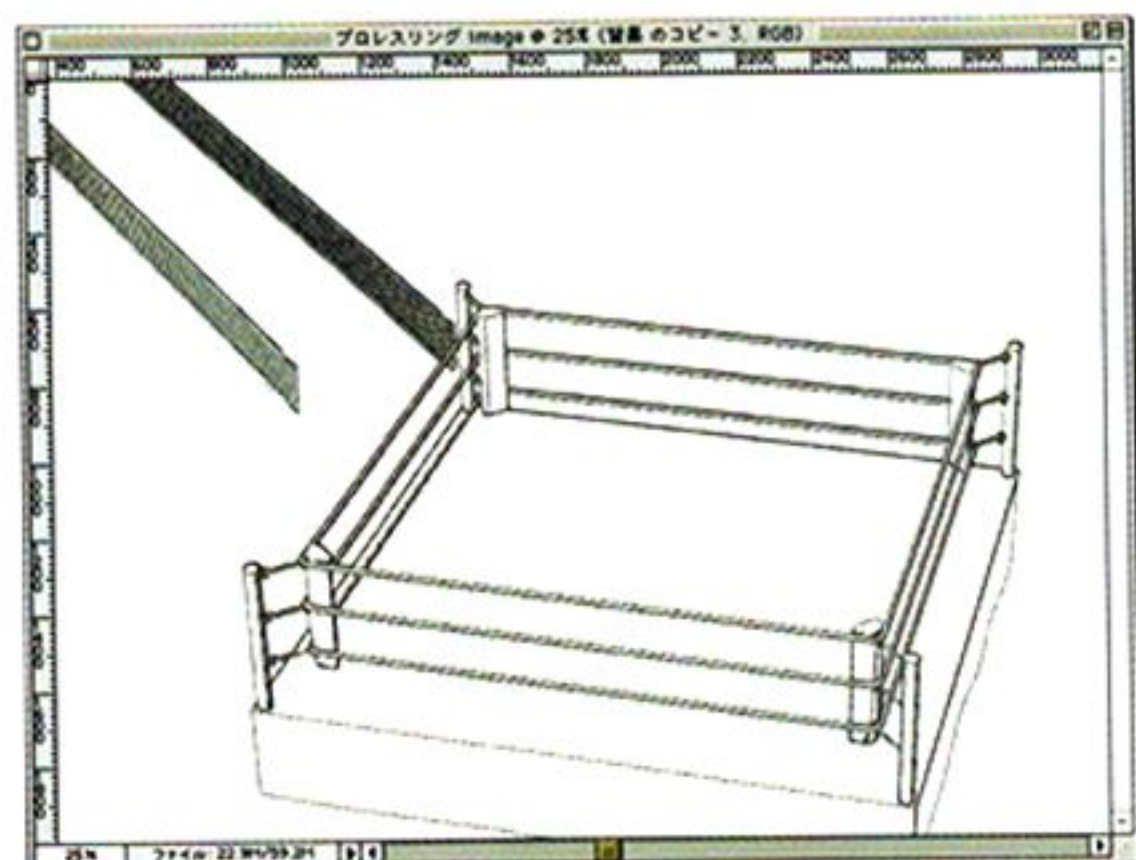



図36　2番目の画像。中くらいのしきい値

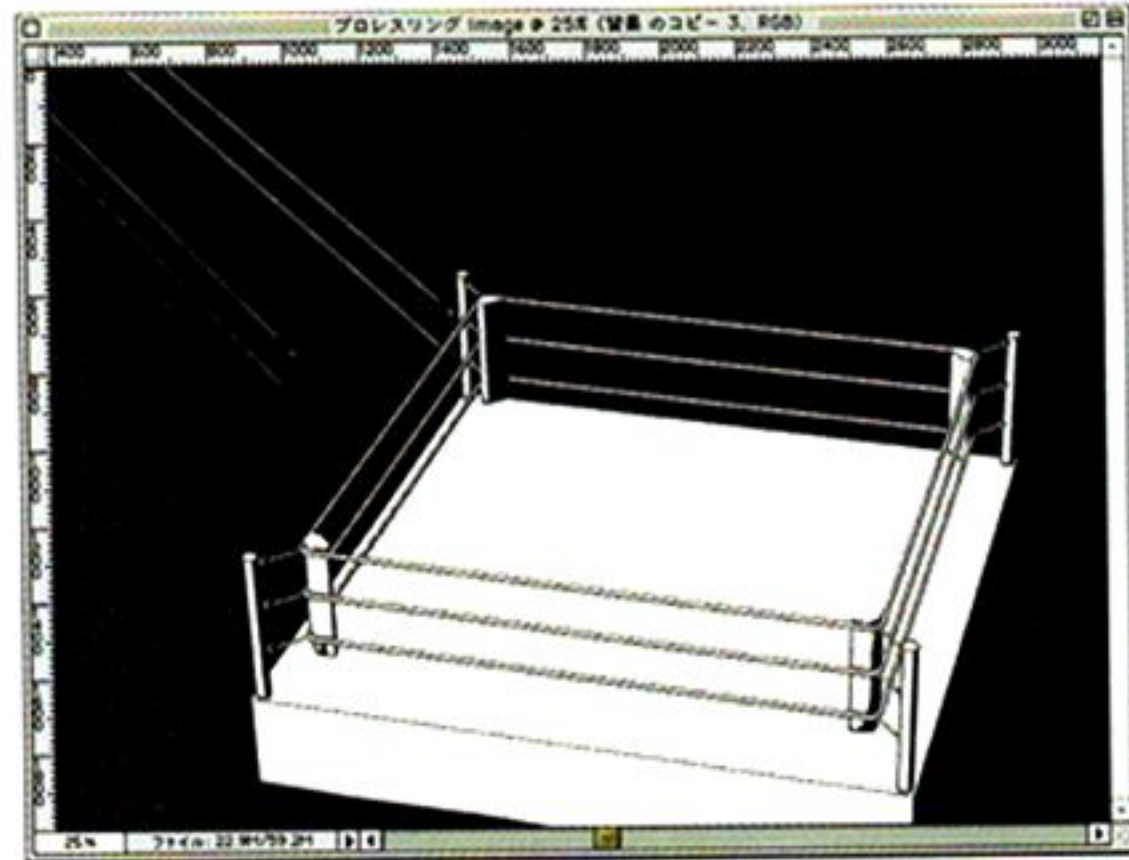



図37　3番目の画像。もっとも黒が多い。3種類の二値画像のしきい値はそれぞれ自分で見つける

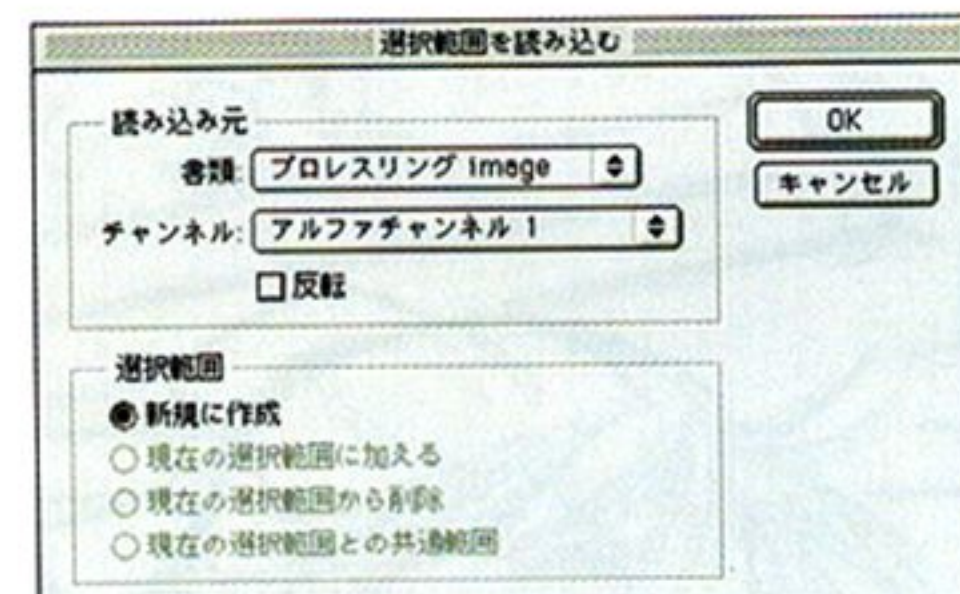



図39　アルファチャンネルからリングのマスクを選択範囲として読み込む

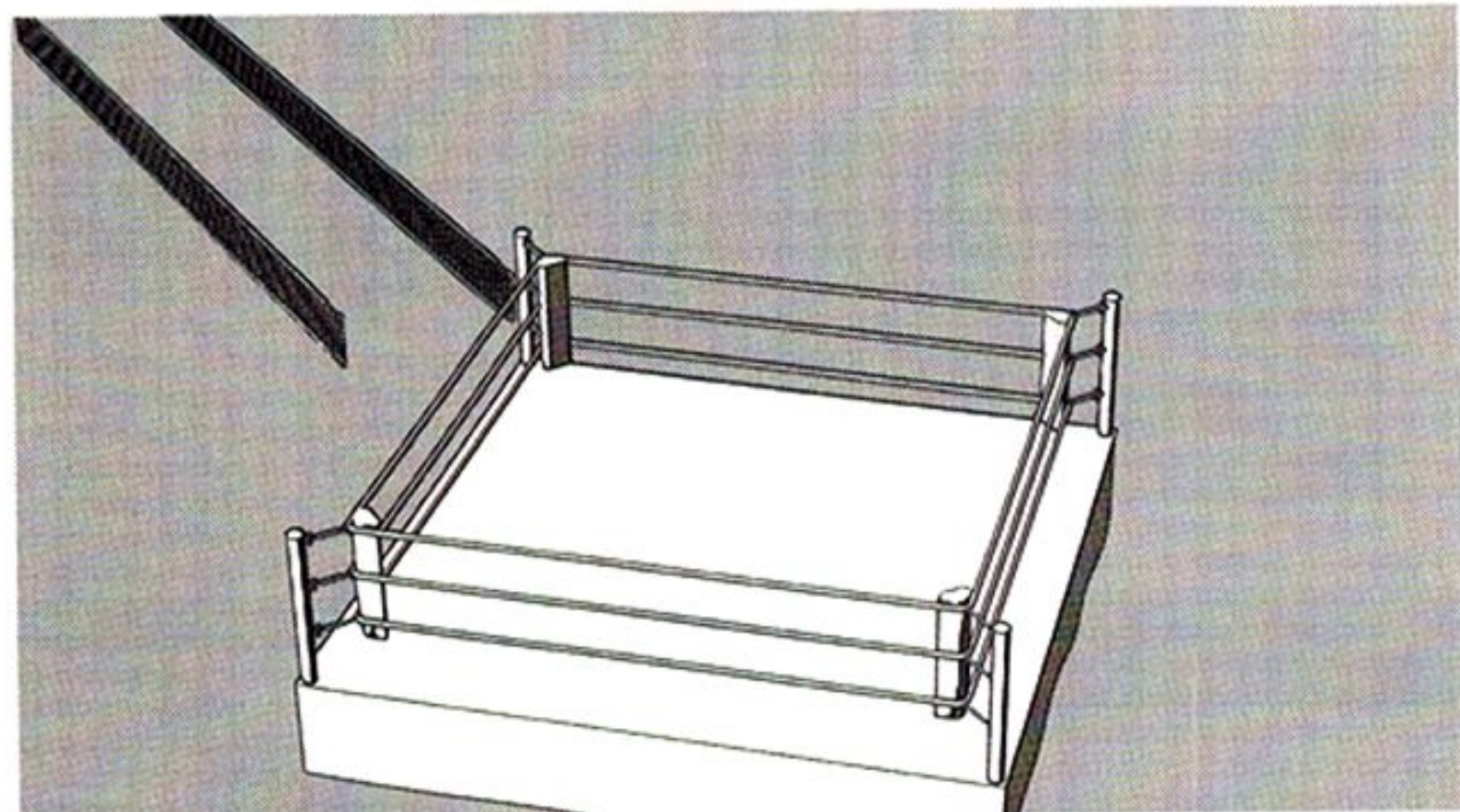



図38　アルファチャンネルの画像と黒の多い3番目の画像から作成した選択範囲を利用してトーンを貼り分けた例

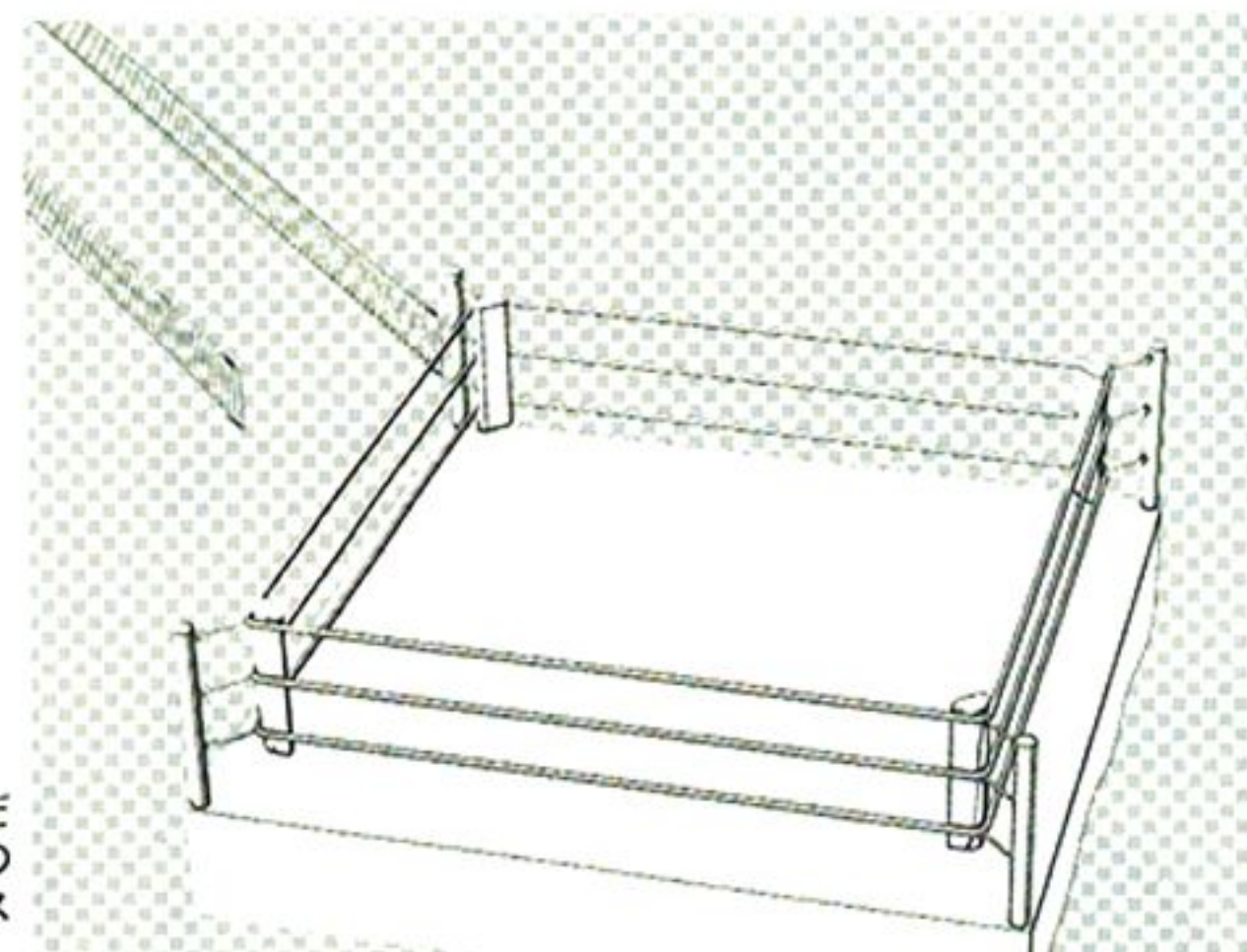



図40　読み込んだ選択範囲を2番目の画像にレイヤーマスクとして加える

図41　リングのレイヤーを原稿にドラッグしたら「乗算」にして「移動」「拡大縮小」で位置あわせ

図42　下のキャラクターがリングの下に隠れる

にそのレイヤーの合成方法を「乗算」にして下にある絵と位置あわせをします（図41）。位置合わせがすんだら合成方法を「通常」に戻し，隠れたリング上の人物と下のコマの人物がはみ出してる部分を削りだします。ここではあとで修正がきくようにリングのレイヤーのレイヤーマスクに対して修正をかけます（修正前図42，修正後図43）。あとはリングと背景のモブが噛みあわない部分をスタンプツールで修正していきます（図44）。

別レイヤーにグレイを塗る 図45

このグレイはあとでアミトーンに変換しますが（プラグインツールのPower Toneを使う場合にも選択範囲として使用できます），この段階で塗っておくのは，次に配置するスピード線・集中線・流線の間の白い抜きを確認しやすくするためです。

塗潰しツールを使うと，線の隙間から色が漏れてしまいますから（図46），あらかじめ鉛筆ツールを使って隙間を塞ぐように描きます（図47/グレイレイヤー以外のレイヤーに描き込まないように）。ペンツールではエッジの色が変わってしまい，塗潰しツールを使ったときに塗り残しになるので，必ず鉛筆ツールを使います。隙間を塞いだら，塗潰しツールのオプション「全てのレイヤーに使用」をチェックして（図48）塗り潰します（図49）。私のやり方だと主線にアンチエイリアスがかかっているので，主線の縁に塗り残しがでないように注意します。それ以外の塗り残した部分は同じく鉛筆ツールで修正します。

スピード線・集中力・流線の作成

アクションマンガにはスピード線（背景の流線）は欠かせません。

よくスピード線には「抜き」が必要だといわれます。漫画雑誌の印刷は原画の線がひとまわり太く（ちょうど線の周りを囲むように）印刷されます。ですから普通に尖った程度の線では周りに贅肉がついてボッタリとしたカッコ悪い線になってスピード感が損なわれます。スピード感のある線を印刷に出すためには，スピード線の先端が印刷には出ないほど細いシャープな線になって消失していなければなりません（プロの原稿を見たことのない人にはイメージしにくいとは思いますが）。

ですから，高品質のレーザープリンタが出ても，スピード線だけは「抜きが出せないからまだまだ使えないなぁ」などといわれてきましたが，そういうセリフを聞くたびに私は「抜きがなければ作ればいいのに」と思ったものでした。

もっとも複数のスピード線を効率よく作成するためには抜きがあるだけでは不十分です。これらの描画にはドローソフト「Expression」が威力を発揮します。

ExpressionはIllustrator同様，ドローソフトです。ドローソフトとしての機能も十分押さえてありますが，図50のように一本のストロークを定義することで，定義したストロークをフリーハンドでさまざまに変形させなが

図43　リングのレイヤーマスクを修正して人物を削りだす

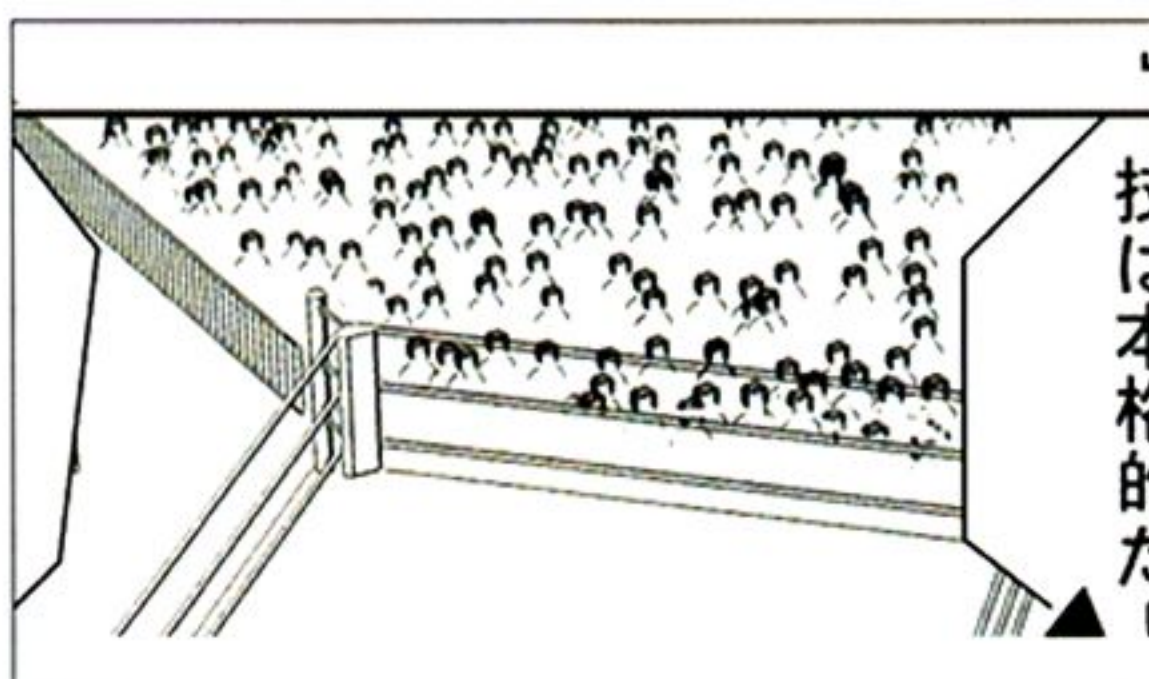



図44　リングと群衆の嚙み合わない部分をスタンプツールで若干修正

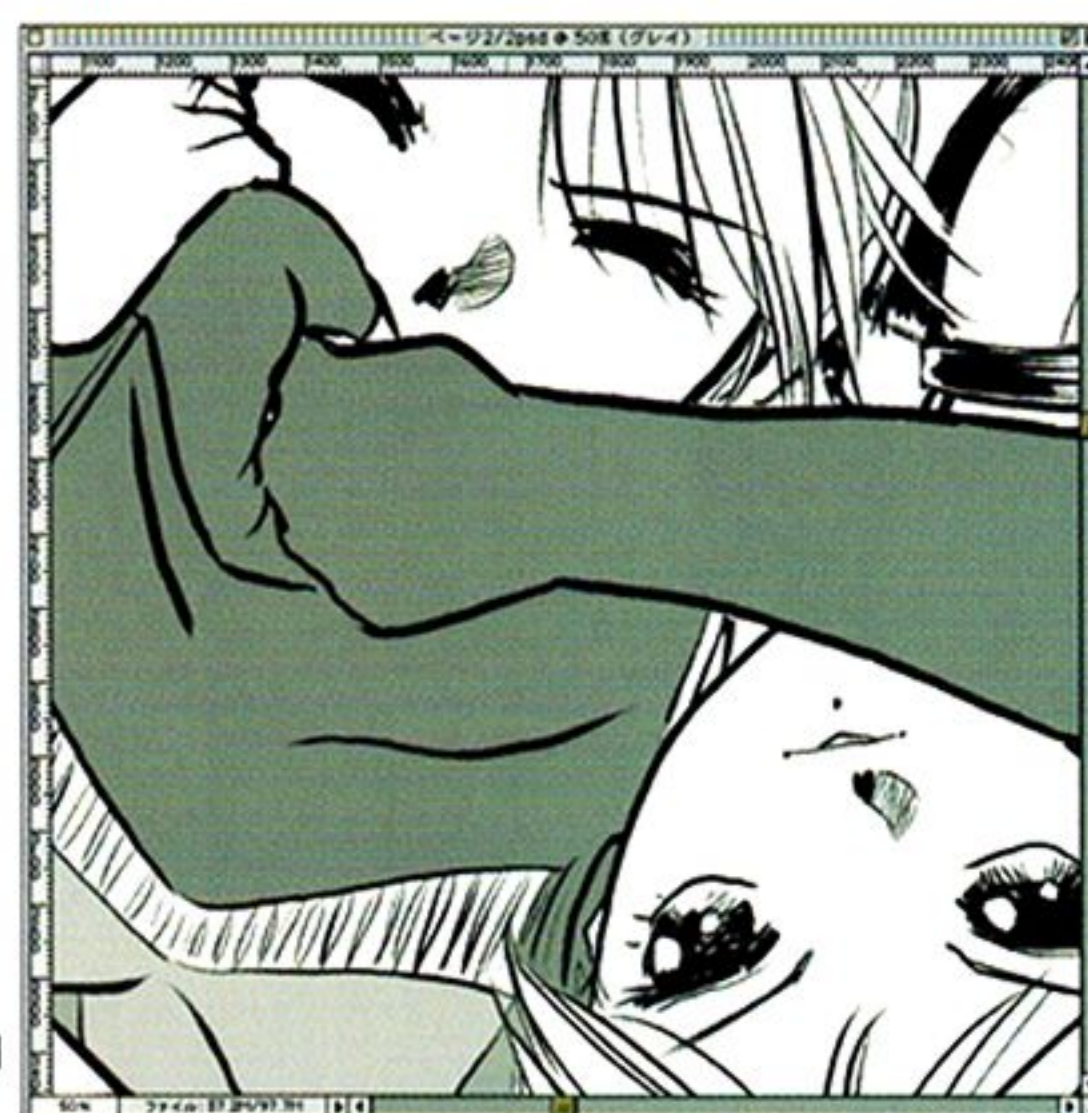
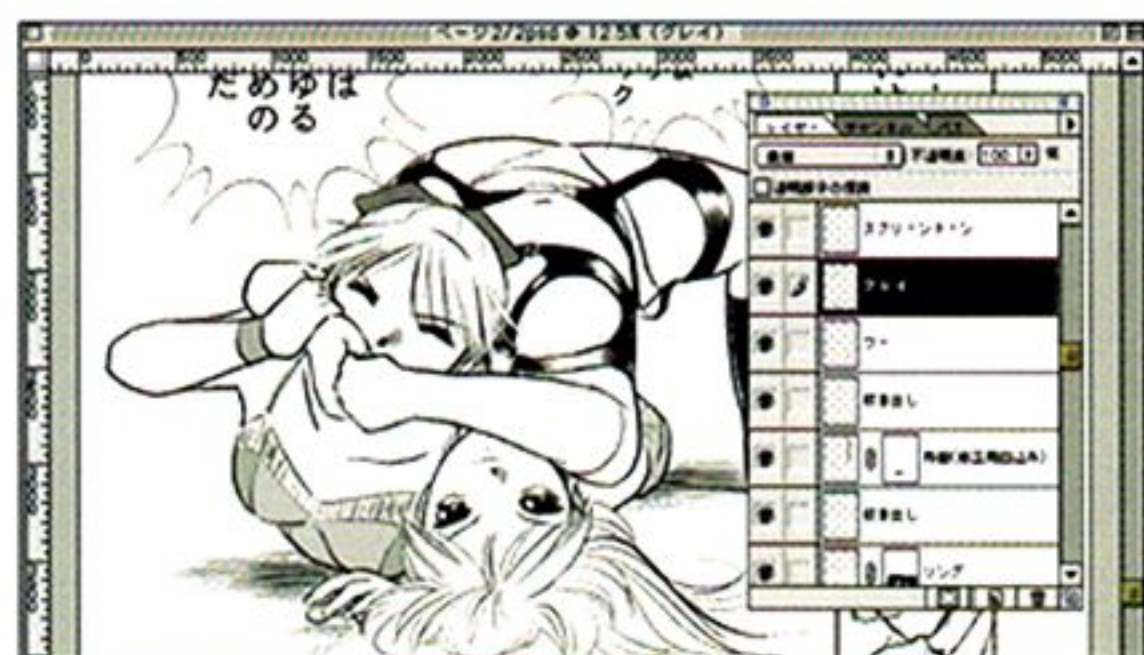



図45　別レイヤーにグレイを塗る

図46　主線の隙間から色が漏れる

図47　主線の隙間を「鉛筆ツール」で埋める。ただしグレイを塗るレイヤーに描く

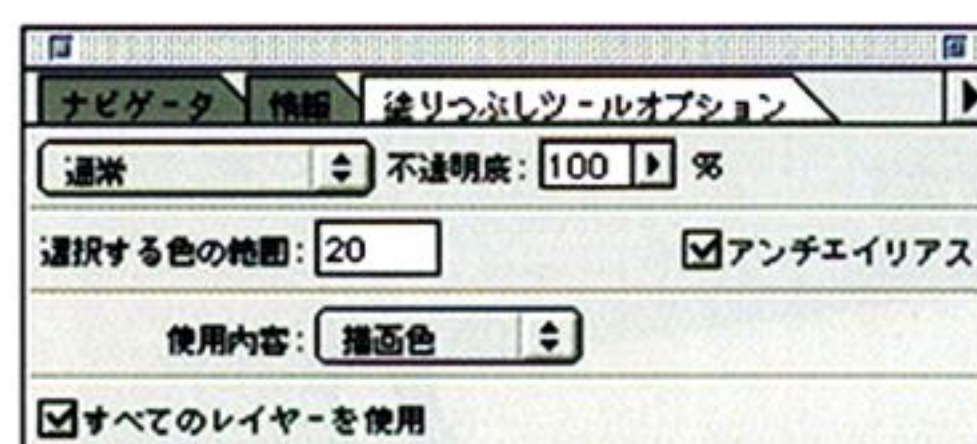



図48　塗り潰しツールのオプション。「全てのレイヤーを使用」をチェック。主線レイヤーとグレイレイヤーのみ表示

図49　色は漏れない

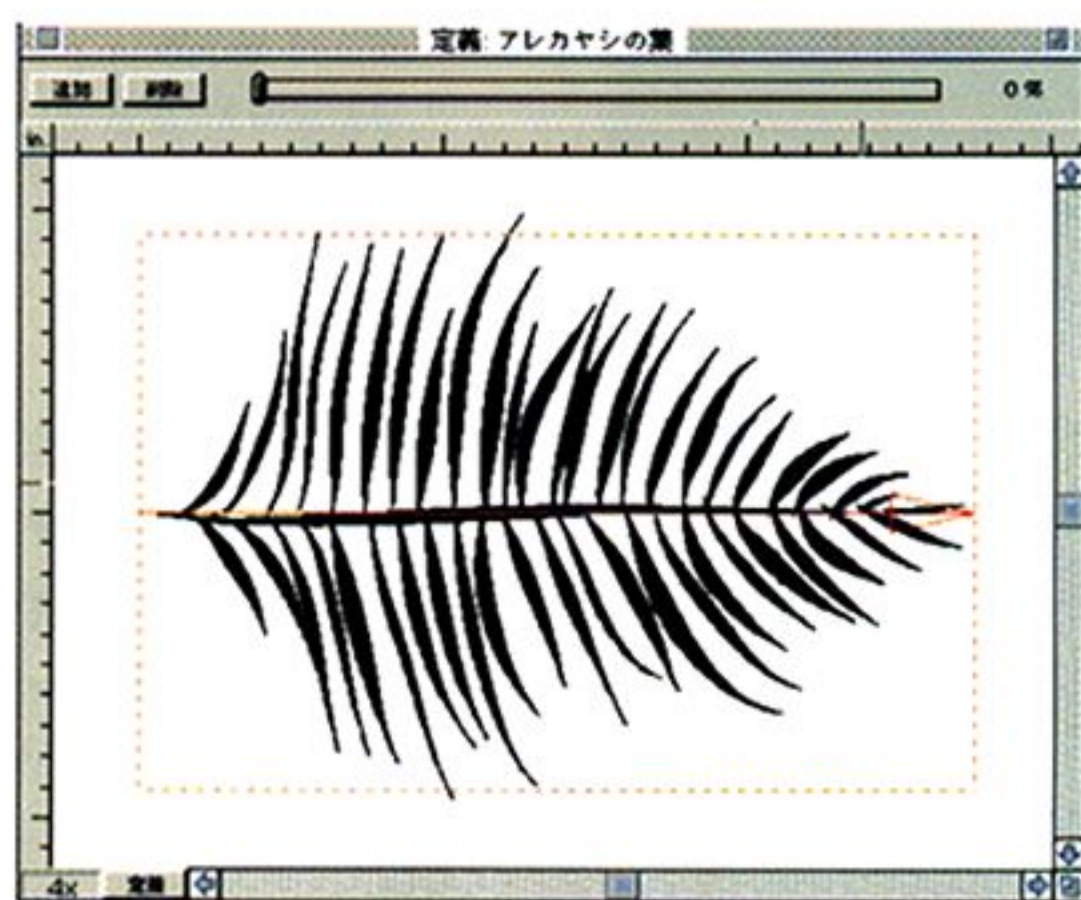



図50　Expressionで葉っぱをひとつのストロークとして定義する

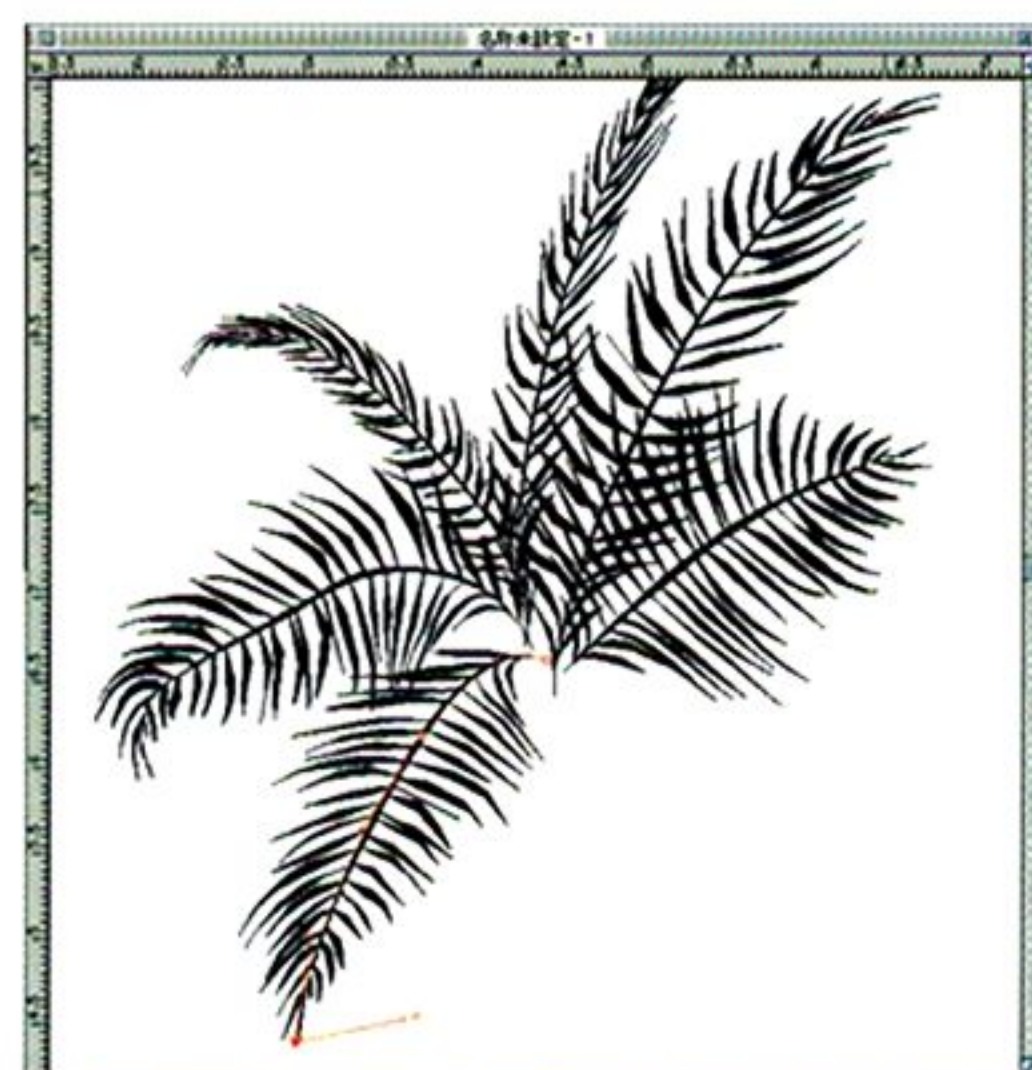



図51　定義された1本のストロークからいろんなバリエーションの葉が描ける

ら（筆圧を感知して太さが変わる）図51のような絵が描けるという手描き感覚のソフトです。このExpressionでスピード線をあらかじめストロークとして定義しておくことで，さまざまなスピード線や集中線をあっという間に作成できます。

作成したスピード線や集中線はEPS形式で保存しておいてからPhotoshopで開き，600dpiグレイスケールのビットマップ画像に変換しています。

まず，1本のスピード線を作成します。図52のような先端がとがったオブジェクトを作成します。単純な三角形でも十分だとは思いますが，一応先端が鋭くなるよう意識しています。オブジェクトの属性は線がなしで塗りが黒から白へのグラデーションになってます。グラデーションは先端部分だけにかかっていて，先端に行くほど白くなっていきます。非常に単純ですが，これがスピード線の「抜き」になります。私の使っているキヤノンのレーザープリンタLBP-750のシリーズのMac用ドライバは非常に細かい誤差拡散モードを持っていて，このモードで出力したときに，先端のグラデーション部分が「スピード線の抜き」となります。手描きのスピード線と比べても遜色がありません。残念ながらWindows用ドライバや他のレーザープリンタについては試したことがないのでどうなるかわかりません。（もっとも「抜き」をアミトーンで表現してはいけないということはありません。自分が納得できればそれでいいはずです）。ちなみにスピード線の抜きだけを取ってみれば，昨今の高解像度インクジェットプリンタなら十分表現できます。今回のマンガ原稿をあえてグレイスケールで表現しているのはこのスピード線の抜きを表現するためだけにあるといってもいいすぎではありません。600dpiの解像度ならスピード線以外のものは枠線でも人物でも描き文字でも白黒二値で構わないはずです。実際，データ入稿におけるモノクロ印刷の場合は線がぼけるのを嫌って白黒二値での入稿が推奨されているようです。

次に，いま作成したオブジェクトを潰して槍のように細長くします（図53）。これをまず1本のストロークとして，ストローク定義ボックスツールで定義します（図54）。そして定義したストロークを使用してペンツールで複数の線を描きます（図55）。線の太さも適当に調節します。基本的にはスピード線の1本1本の線の間隔はランダムに手作業で並べるのですが，Expressionの整列機能（図56）を併用すれば効率よく作業できます。

気に入った並びのスピード線ができたらそれを全選択し，グループ化して90度回転させ，今度はその複数の線をさらにひとつのストロークとして定義してしまうのです（図57）。今回は反復ツールも使用しています（図58）。ストロークが長くなれば反復ツールで定義した部分が繰り返し出現することになります。反復を選択した時点でストローク定義ボックスの端から端までがデフォルトで選択されていますが，この状態でOKです。ストローク定義ボックスの端と端がスピード線（集中線）の継ぎ目になりますので，その間隔にも注意します。

そして定義ボタンを押して，ストロークのデフォルトの太さ（これがスピード線・集中線の「長さ」になる）を最大（256）に設定し，適当な名前をつけ

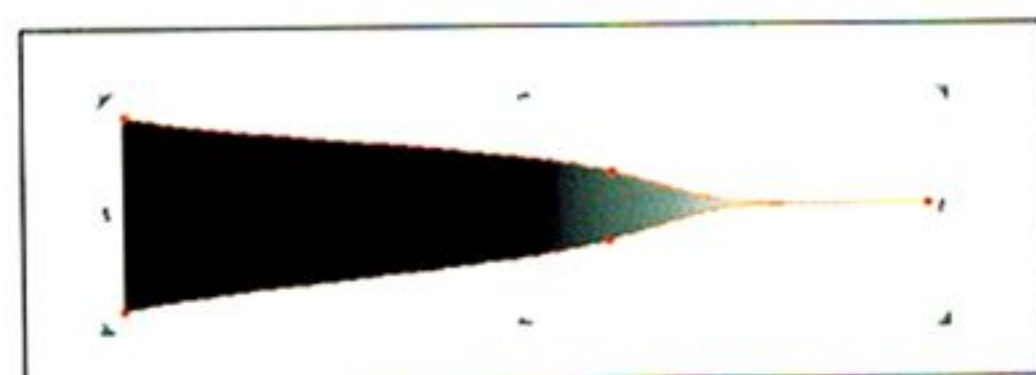



図52　スピード線の基本形。先端の尖った部分にのみ黒から白へグラデーションがかかっている

図53　四隅のハンドル操作で，縦に潰して細長い槍にする

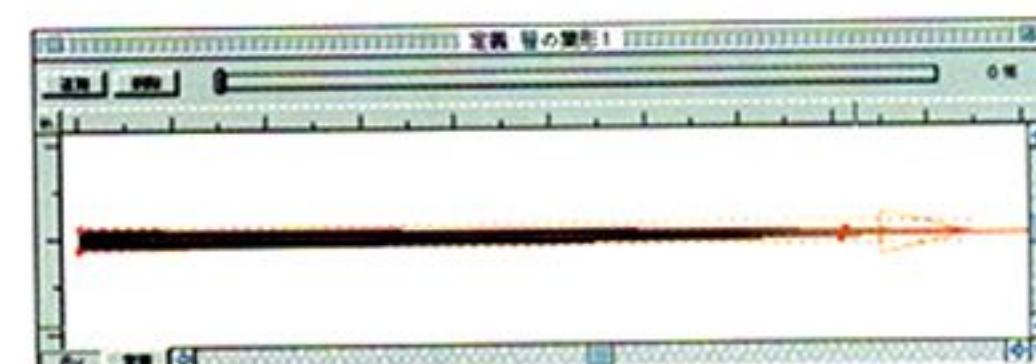



図54　まずこの槍を1本のストロークとして定義

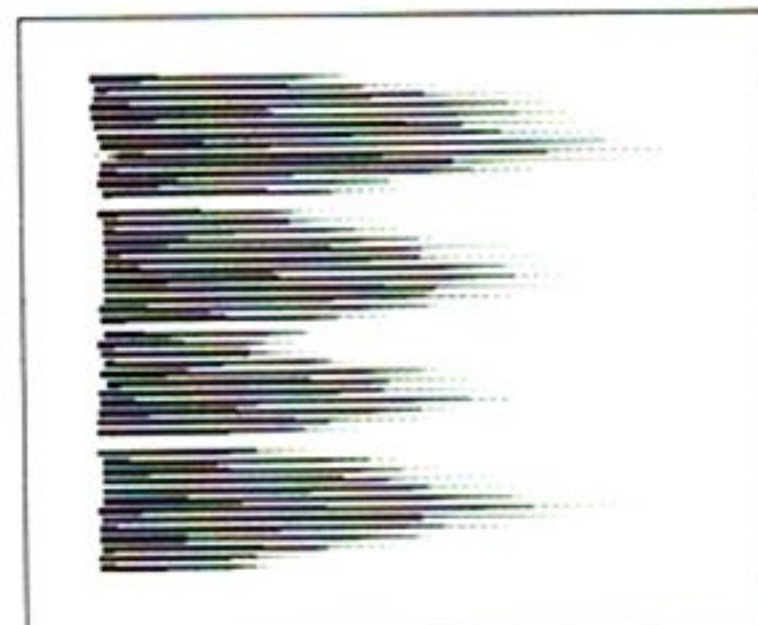



図55　定義した槍を好みのスピード線となるように配置

図56　Expressionの整列機能を利用すれば効率よく配置できる

図59　定義されたスピード線はストロークとしてブラウザに登録される

図57　このように複数のスピード線を1本のストロークとして定義

図58　反復ツールで指定した範囲は1本のストロークのなかで繰り返し描画される

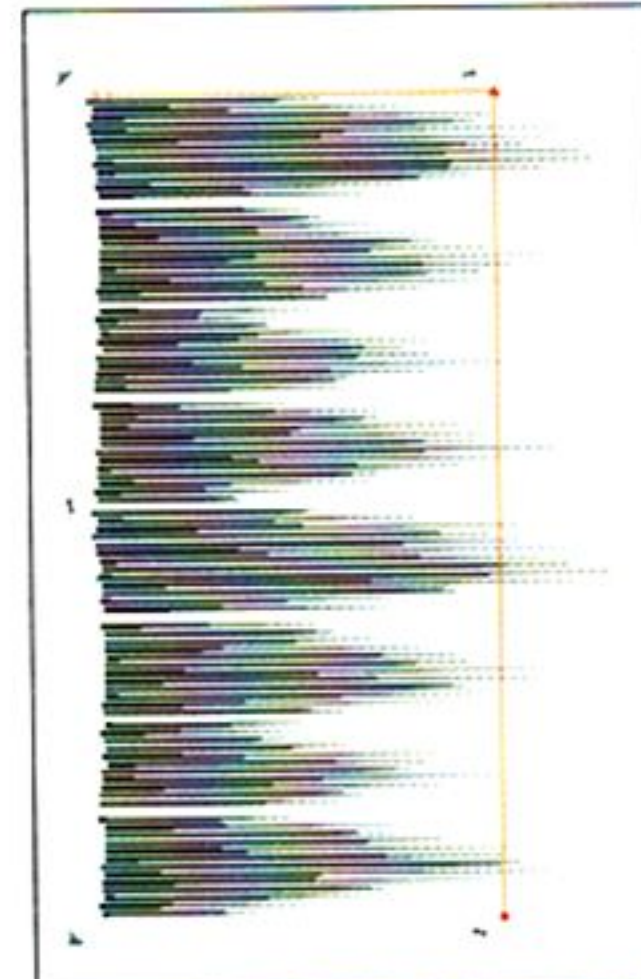



図60　定義した「スピード線」ストロークで下から上に線を引き，ストロークの太さを調節することでスピード線の長さを調整する

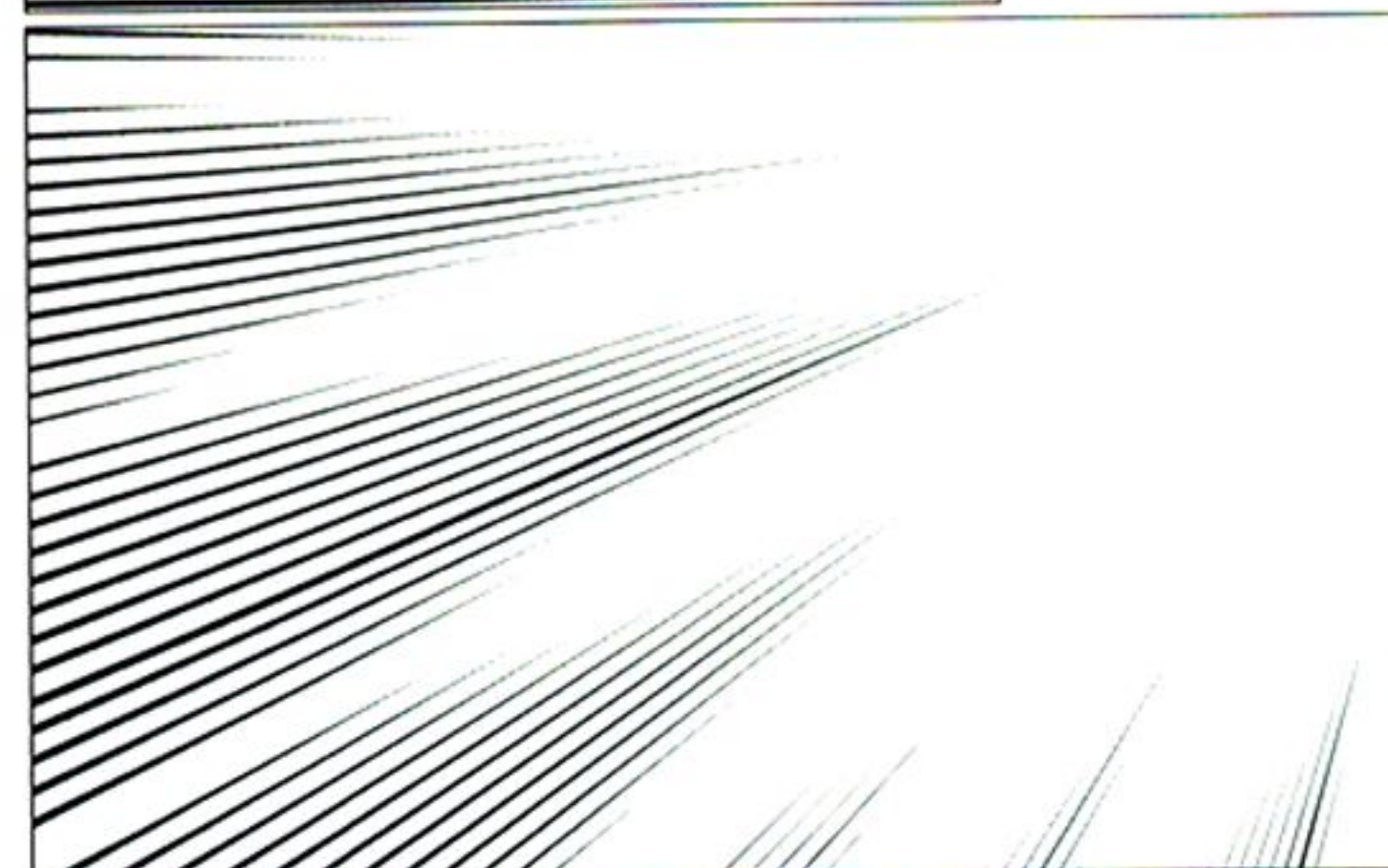
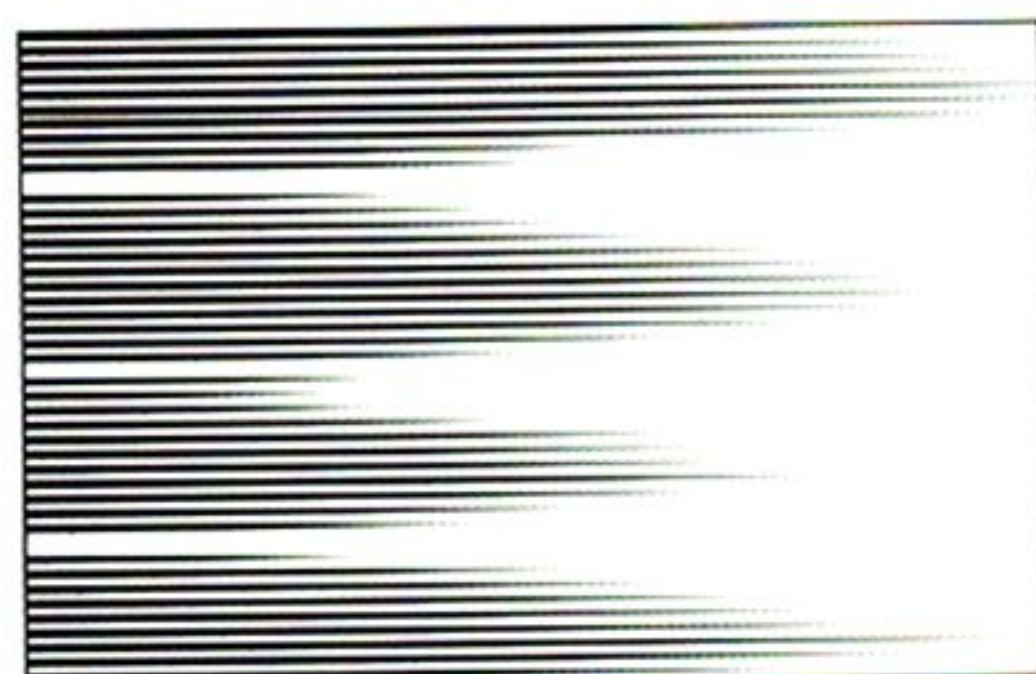



図63・64　ビットマップデータに変換したもの。綺麗な集中線になっている

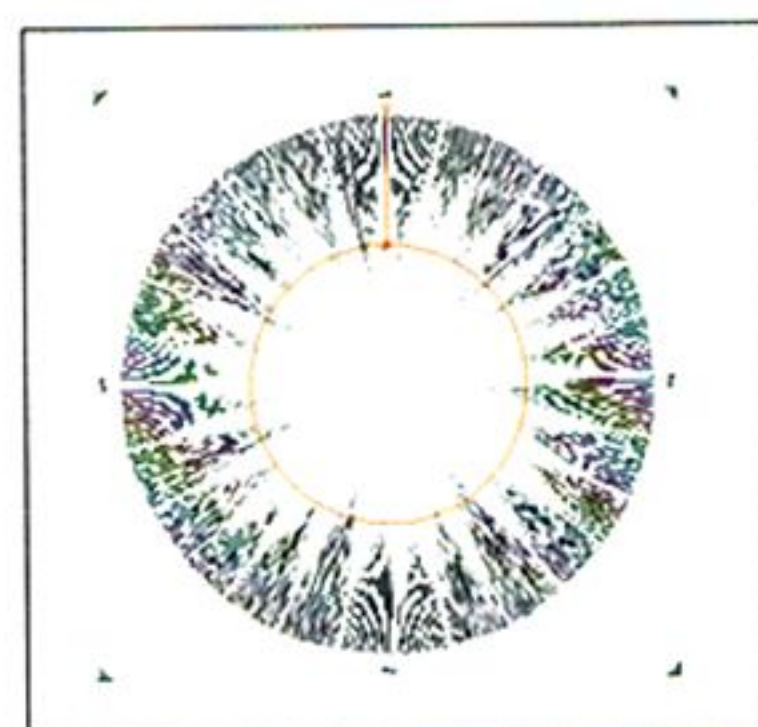



図61　同じ「スピード線」ストロークで「円」を描くと集中線になる

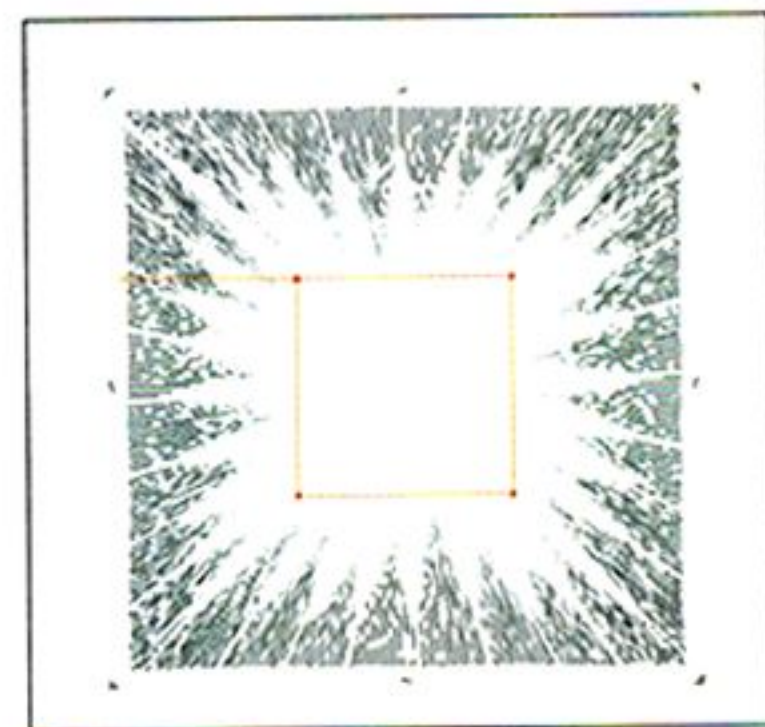



図62　正方形を描いても違った形の集中線になる

て保存するとほかのストロークと同じようにブラウザに登録されます（図59）。

このようにしてスピード線をあらかじめ定義しておけば，実際に描画するときは，図60のようにペンツールで下から上に垂直線を引いてストロークの太さ（スピード線の長さ）を調節するだけで済みます。

集中線を描くときは同じスピード線のストロークを使用し，楕円ツールでシフトキーを押しながら正円を描いて，同様にストロークの太さを調節するだけです（図61）。四角形ツールで正方形を描くと，微妙に違った形の集中線になります（図62）。（注：Expressionのプレビューの精度はそんなに高くないので，ギザギザの線に見えますが，ビットマップデータにコンバートすれば綺麗な集中線であることがわかります。図63・64）

ここで気をつけることは，楕円や長方形をいきなり描いてはいけないということです。スピード線はストロークに対して単純に直角であるため，いきなり楕円を描こうとすると図65のようになってしまいます。楕円や長方形に加工したいときは，必ず先に正円か正方形を描いてから四隅のハンドルを操作して加工しなくてはなりません。そのほかにもハンドルの長さと拡大率を調節することによって，ひとつのストロークからいろんなバリエーションの集中線が描けます。図66（部分拡大図図67）が円の集中線のバリエーションで図68（部分拡大図図69）が四角形の集中線のバリエーションです。

垂直・水平に走るスピード線を描く場合にも注意が必要です。そのままPhotoshopデータにコンバートすると，ピクセルのギザギザが多少目立ってしまうので（図70），いったんExpression上で30度程傾けてから（図71）コンバートし（図72），Photoshop上で再び回転させてから元の角度に戻すようにしたほうがいいでしょう（図73）。

そのほかにもいろいろな加工ができます。通常のスピード線のハンドルを斜めに傾けただけでも違ったスピード線を得ることができますが，面白いのは円を描画してできた集中線のハンドルを傾けると図74のような渦巻き形の流線ができることです。これもいろんな使い道があるでしょう。

加工の方法はまだまだあります。パスを部分的に切り取ったり，切り取った部分のハンドルの長さを変えてみたりすることで同じストロークからさらにバリエーションを増やすことができます（図75）。

パースペクティブツール（図76）を利用すればバリエーションはさらに広がります。通常の集中線をパースペクティブツールで加工すると図77のようになります。ハンドルを傾けると図78のようにすることも可能です。こ

図65 楕円をいきなり描くとこうなる。楕円は正円を描いてから四隅のハンドルを動かして加工する

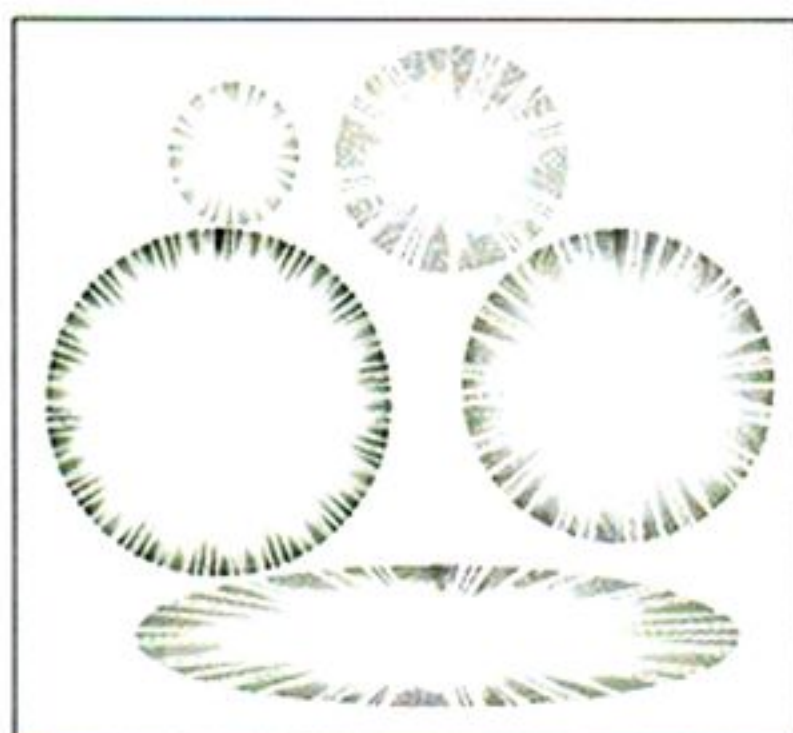



図66 ひとつのストロークで描ける円のバリエーション

図67 円のバリエーションから切り取った集中線。同じストロークからできてるようには見えない

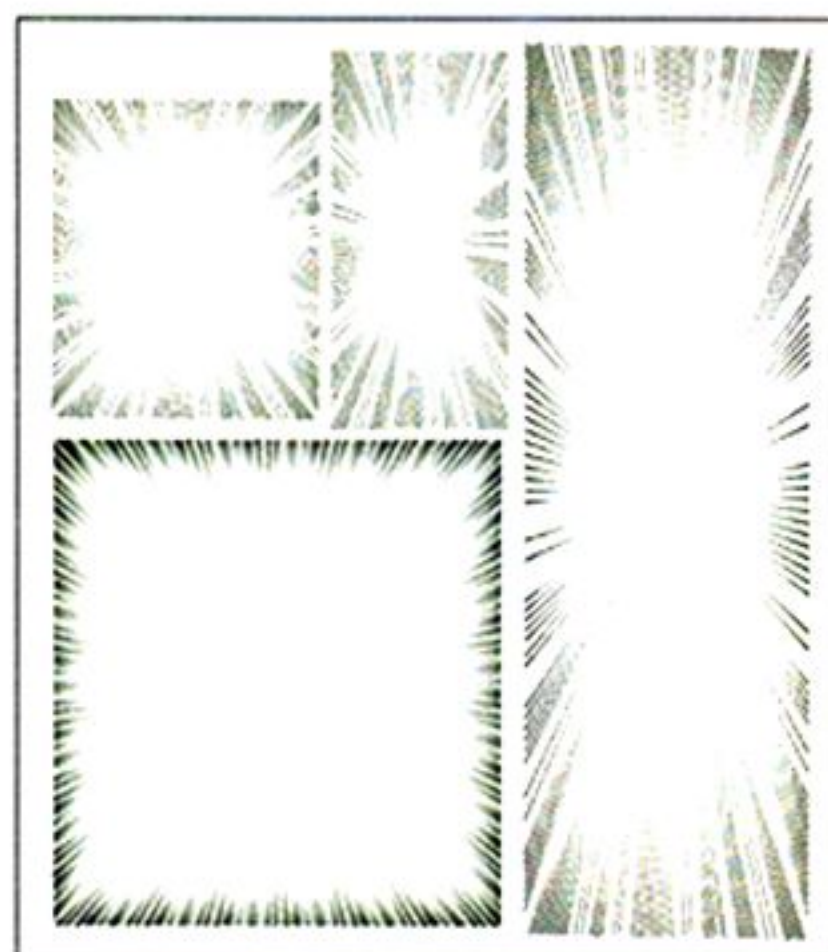



図68 こちらは同じストロークの正方形バリエーション

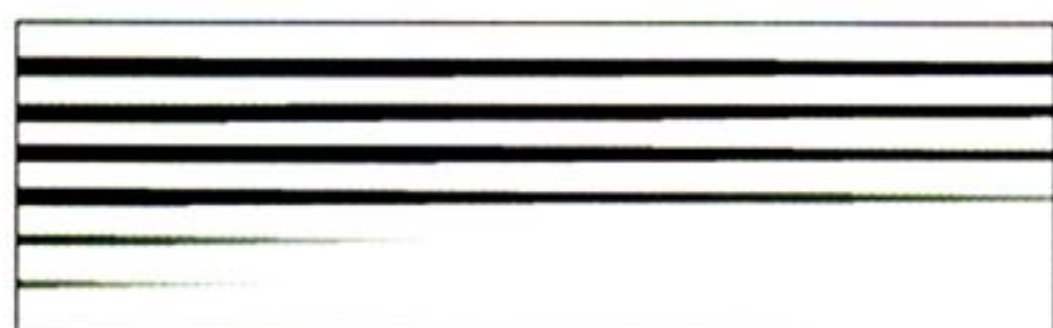



図70 水平垂直のスピード線はそのままビットマップデータにすると多少ギザギザが目立つ

図71 水平垂直のスピード線はEcpression上で30度程傾ける

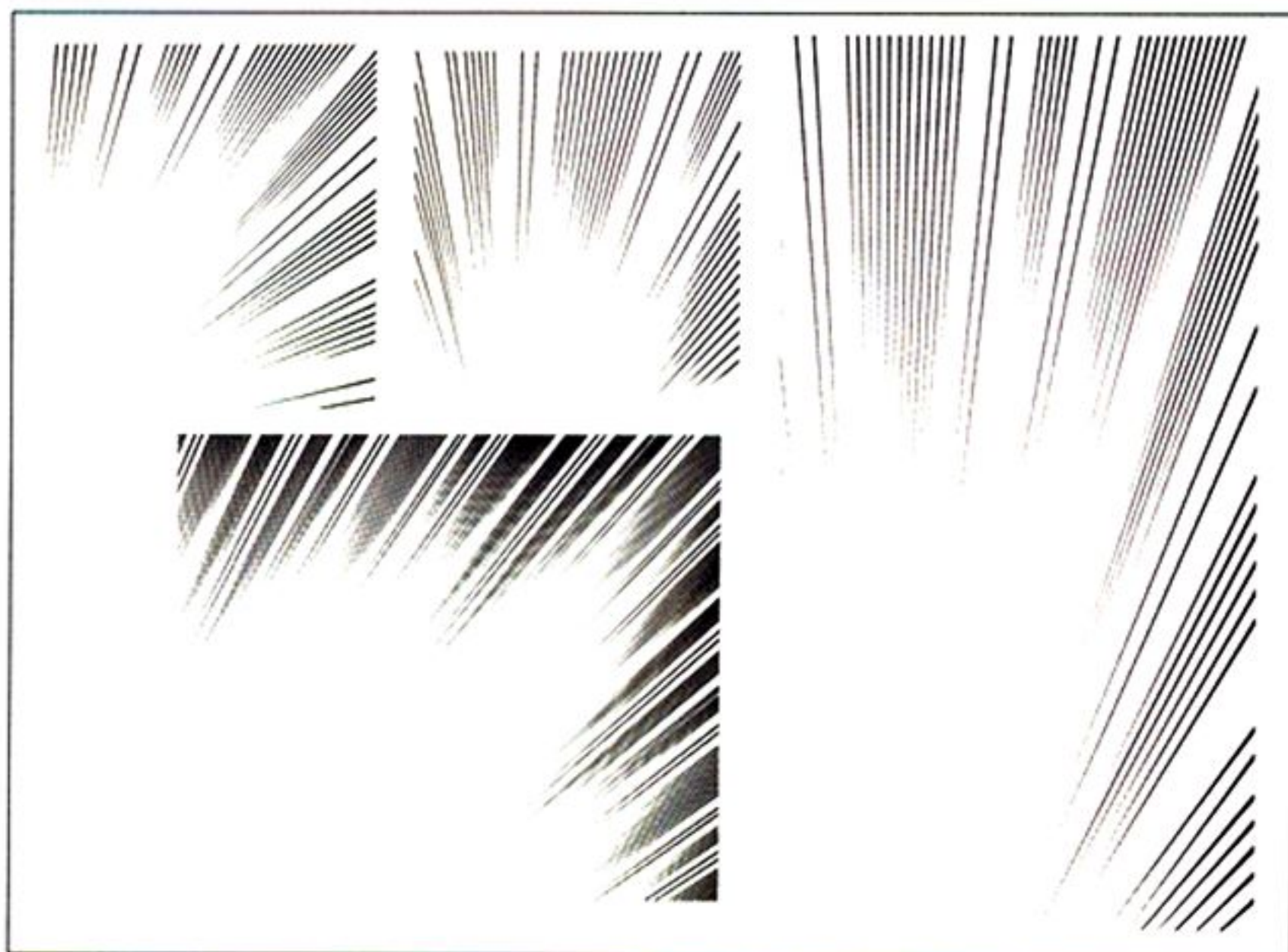



図69 正方形のバリエーションから切り取った集中線。円と正方形だけでいろんなバリエーションが得られることがわかる

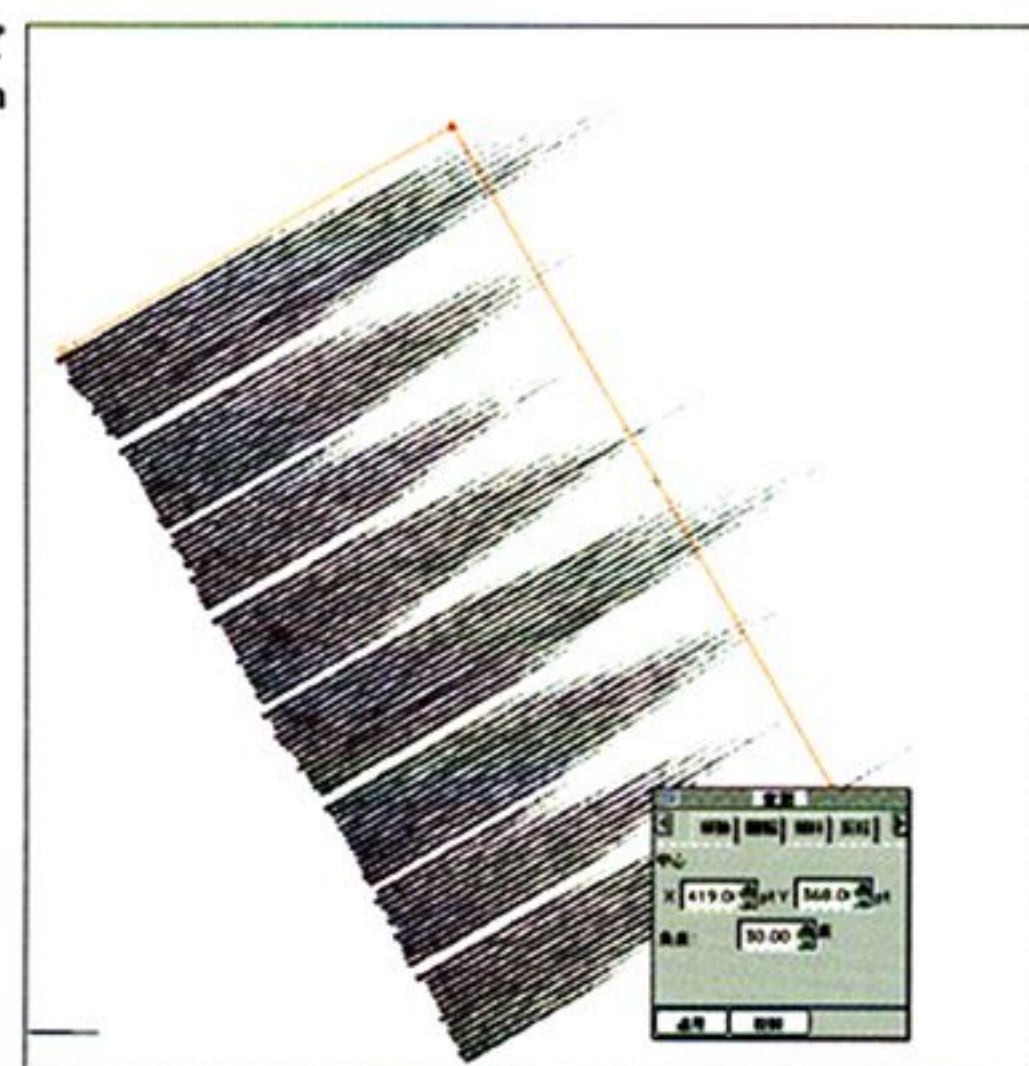



図72 傾けた状態でPhotoshopデータに変換する

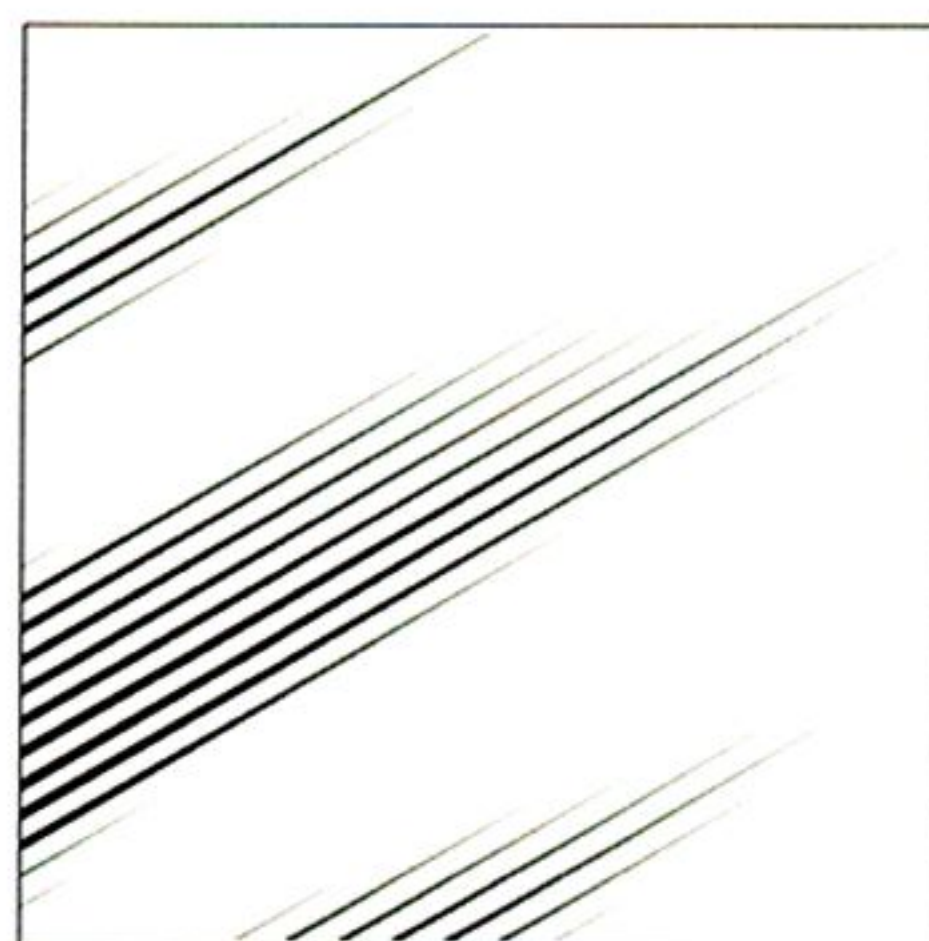



図73 Photoshop上傾きを水平に戻すとギザギザは出ない

のように，たったひとつのパターンからアイデア次第でいろんなスピード線表現が可能であることがわかると思います。

主人公の独白などによく用いられるトゲトゲのある吹き出しを私たちの仲間内では「ウニ」などと呼んでいますが，このウニを描くのには意外と時間がかかるものです。集中線と同じ原理の応用でこの「ウニ」も簡単に作成できます。

まず，スピード線と同じ要領で今度は左右両端が抜けている（両端にグラデーションがかかっている）短めの線を描き（図79），これをストローク定義して，さらにこのストロークを並べたものをこれも同じ要領で並べてストローク定義します（図80）。定義したストロークで円を描けば，あっさりとウニができあがります（図81）。使用する状況にあわせて加工しますが（図82），一部を切り取って使うことの多い集中線と違って，ほぼ丸ごと使用するウニはあまり変化のつけようがありません。あまりスマートとはいえませんが，適当にポイントを増やして形を少し崩してみるなどして加工するのがいいでしょう（図83）。

回転を表現するための楕円形の流線も同じ要領です。しかし，両抜きのスピード線を定義して楕円を描くとグラデーションが思った形で反映されないため（図84），抜きのグラデは諦めて，形状のみ両端が尖った黒ベタの線にしました。抜きが必要なときはビットマップに変換したあと，Photoshop上で消しゴムツールのエアブラシでもかけることにします。

スピード線もそうですが，あとでPhotoshopに持っていったときに線と

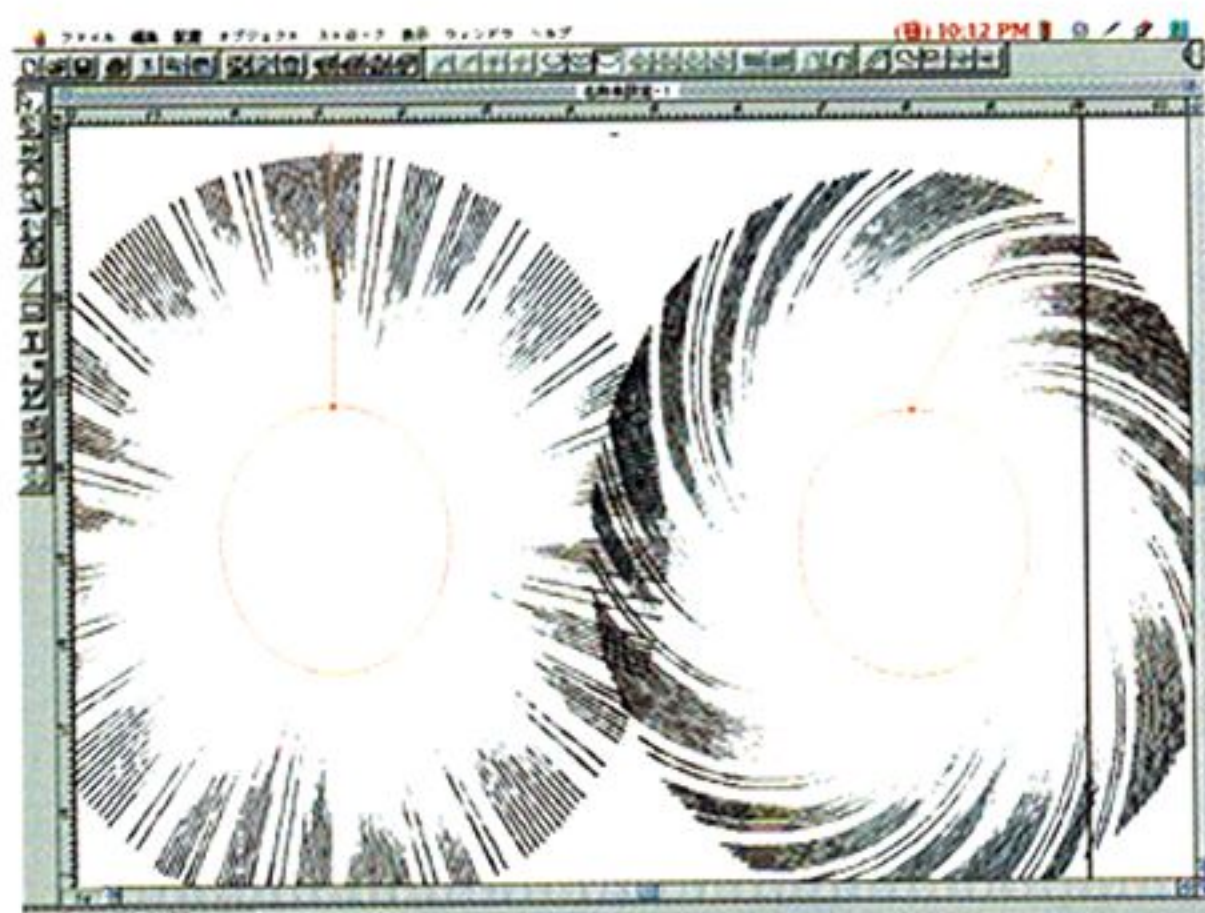



図74　円の集中線のハンドルを傾けると渦巻き状の流線になる

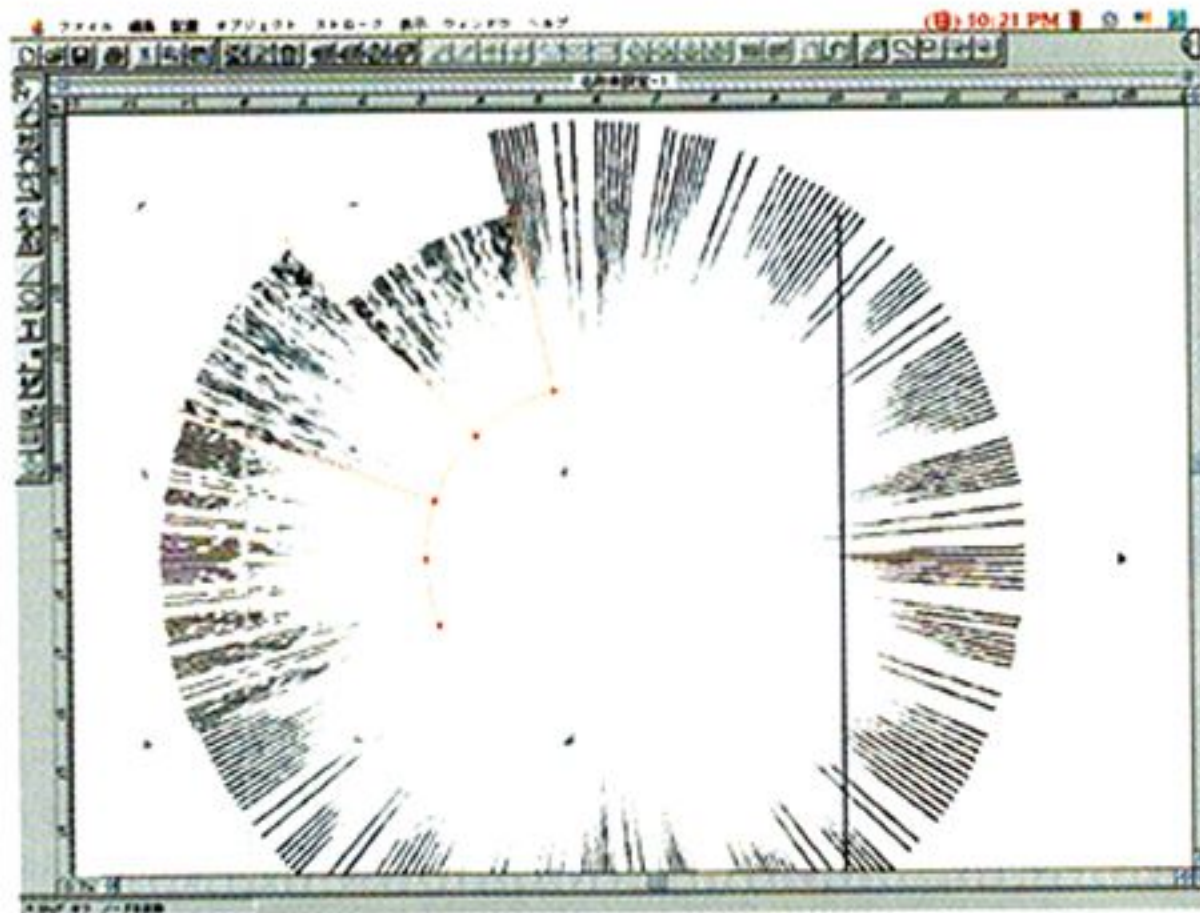



図75　パスを途中で切断したり，切断した部分のハンドルの長さを変えることで，さらにバリエーションが得られる

図76　Expressionのパースペクティブツール

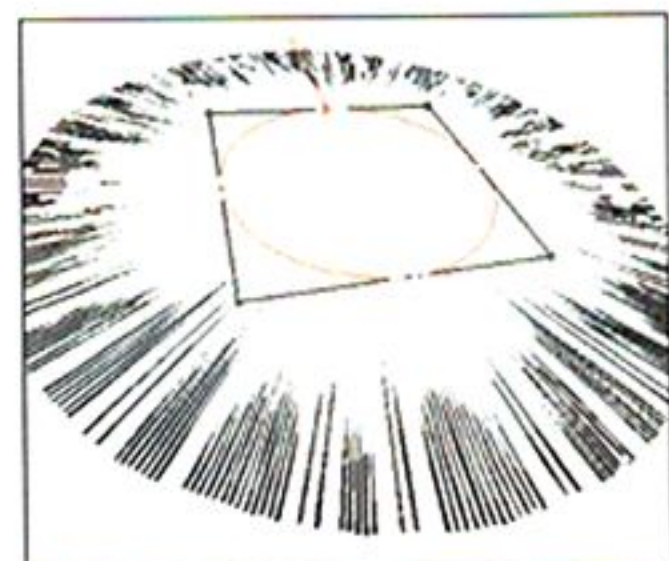



図77　パースをつけることでさらにバリエーションが広がる

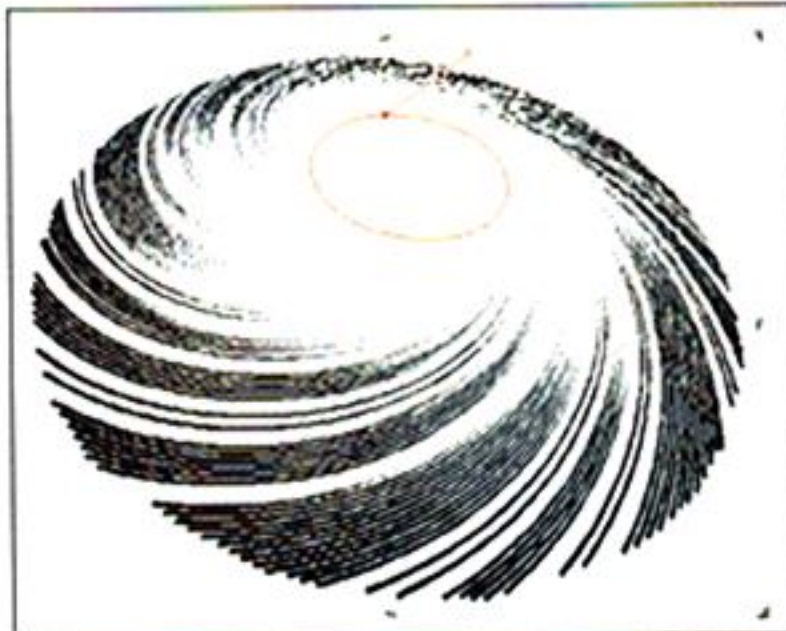



図78　ストロークのハンドルを倒すとパースのついた流線に

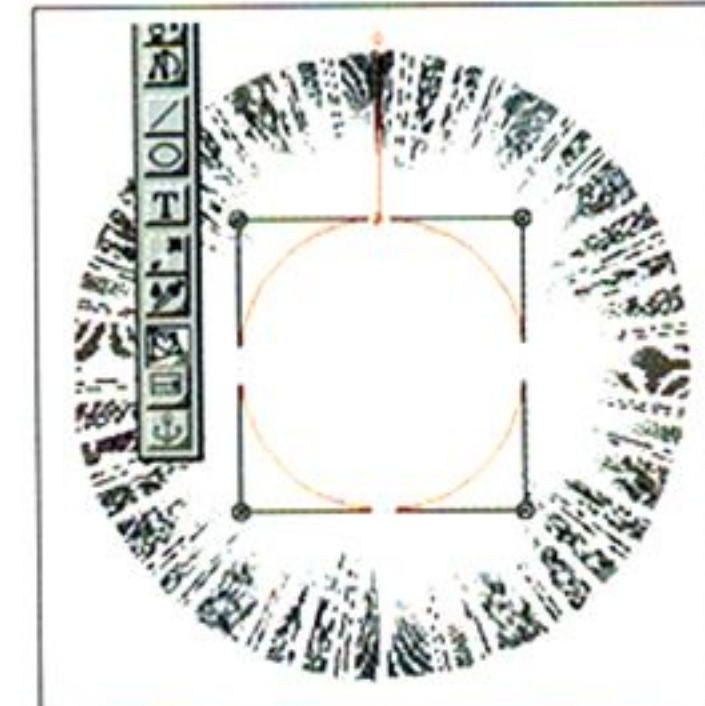
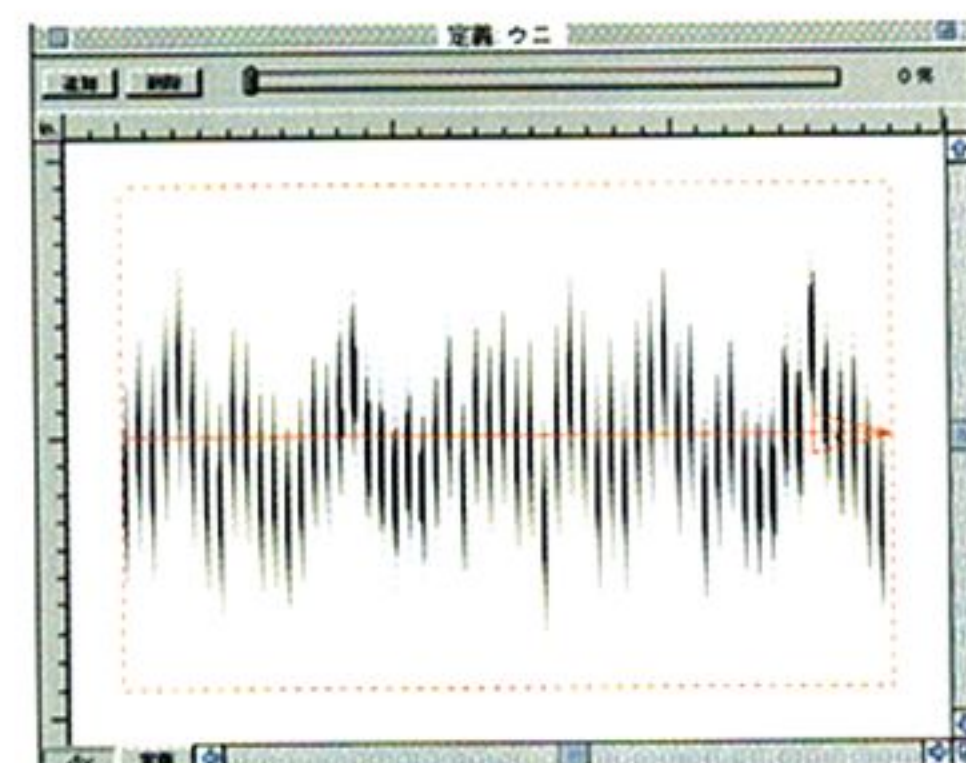



図80　複数の槍を並べてストローク定義

図79　両抜きで短めの槍を定義

図81　円を描くだけでウニができる

図82　四隅のハンドルで楕円にする

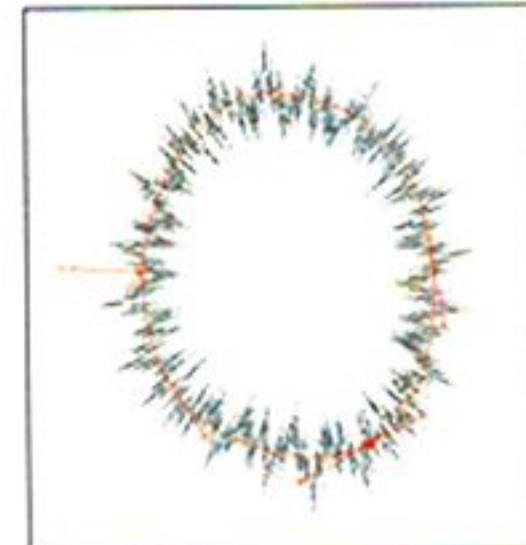



図83　パスに変換して適当に形を崩す。最初からペンツールで丸を描いてもこのくらいにはなる

図85　線と線の間をあらかじめ白く塗ってある線を定義

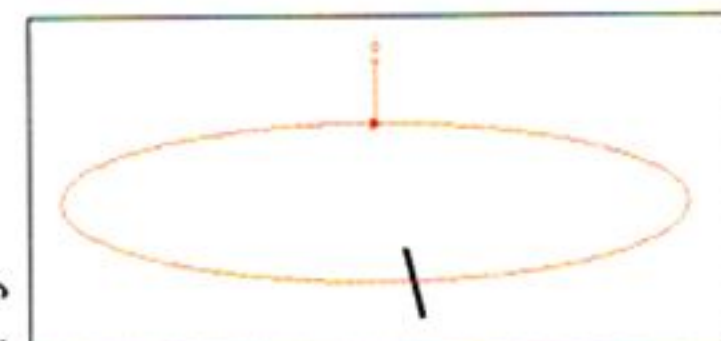



図84　両端にグラデのかかったストロークのはずが……

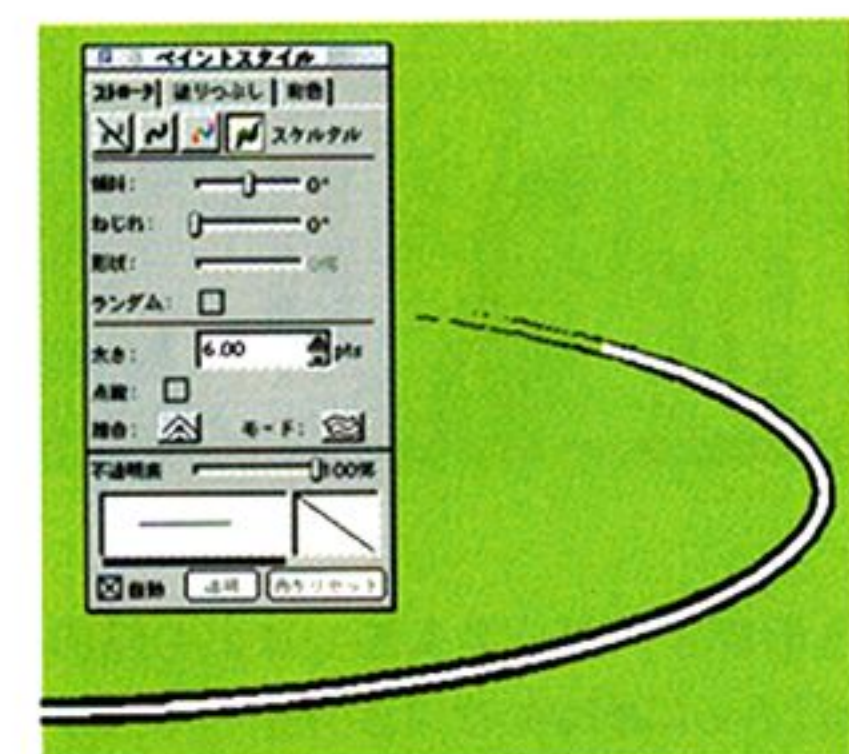



図86　ソーセージモード。ストロークに対し直角に太さを反映

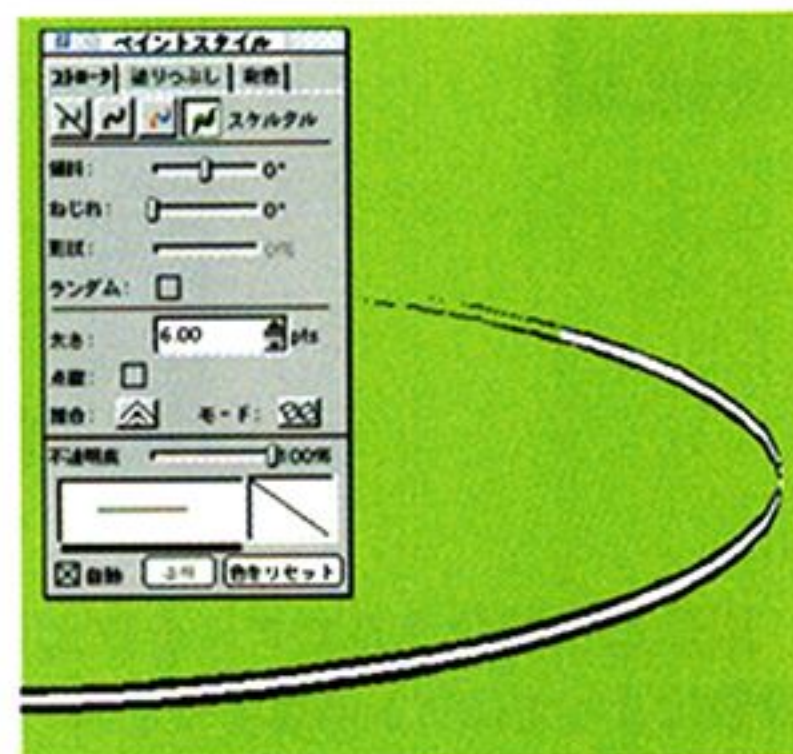



図87　リボンモード。太さは常に一定の方向に向いている

線の間の細い隙間を白く塗り潰して修正するのは面倒です。そこであとの修正が楽になるように，あらかじめ間を白く抜いた線をもうひとつ別に定義してみました（図85）。この2つを使い分けて回転表現の流線を描いていきます。ストロークには標準のソーセージモード（太さをストロークに対して直角に反映）とリボンモード（太さが常に一定方向。Illustratorでカリグラフィと表現されているもの）があります。マンガではどちらも使う場合がありますから状況に応じて使い分けるといいと思います（図86・87）。

この白抜きのついた線で楕円を描き（綺麗さを求めるなら正円を描いて四隅のハンドル操作で楕円にするのがよい），それを上に向かってコピーしていきます（図88）。この上下関係はあとで加工するたびに変わりますから，できあがったあとで点検するクセをつける必要があります。そしてコピーした一部を選択して白抜きのないシングルのストロークに変え，楕円を拡大/縮小したりして形を流線に見えるように配置します（図89）。

すべての流線が楕円上の同じ位置で揃って抜けているのは機械的で不自然ですから，抜ける部分を適当にずらす必要があります。方法としては「パスの開始点を変更する」，「パスを切断して2つに分ける」などがあります。パスの開始点を変更するには，まず「ノード追加」ツールで新たな開始点となるポイントを追加し（図90），「始点を変更」ツールでいま追加したポイントを選択します。図91で下になっているほうのシングルの線が図92のように変化しています。パスを切断すると図93のようになります。

できあがった流線はEPS形式で保存し，Photoshop上で開きます。設定は600dpiのグレイスケールでアンチエイリアスをONにしています（図94）。レイヤーは人物よりは上で，セリフ・吹き出し・枠線・書き文字よりは下になるように配置しています。あらかじめ白く抜いておいた部分は修正する必要がないので（図95）あとの修正が楽です。流線を傾け（図96），人物の後ろに回り込んでいるところは修正します（図97）。修正はあとでやり直しがきくように，レイヤーマスクに対して行っています。こうしておけば配置を変更することも楽にできます（図98）。

スピード線にも同様に白抜きをつけてみました（図99）。この白抜きのついた線を取り交ぜてランダムに並べたスピード線を作成し，すでに述べた要領で集中線を描きます。それをEPSで保存し，Photoshopデータに変換して原稿上に配置します（図100）。抜きのある集中線を人物の上に配置すると，先端のグラデーションで白くなった部分が浮き上がってしまいますから（図101），レイヤーの合成方法を「乗算」に変えます。しかし，そうすると今度は線と線の間の修正用の白まで消えてしまいます（図102）。ですからもうひとつ別のレイヤーを設けてそこに修正用の白を新たに描画するしかありません。修正用の白はそのための選択範囲として使用します。

図88 下から上にコピー

図89 ストロークの一部を別定義のシングルの線に変え，大きさ・位置を調整

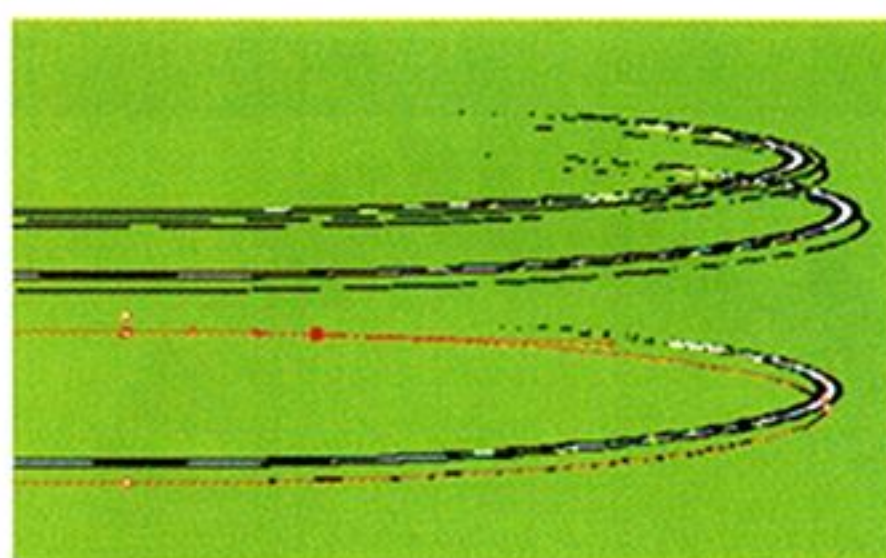



図90 ポイントを追加して，そのポイントを始点に

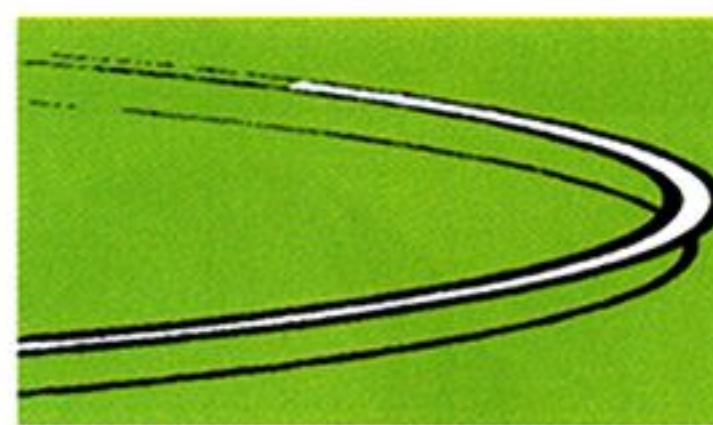



図91 始点変更前（下のシングルの線）

図92 始点変更後。ストロークの先端がずれている

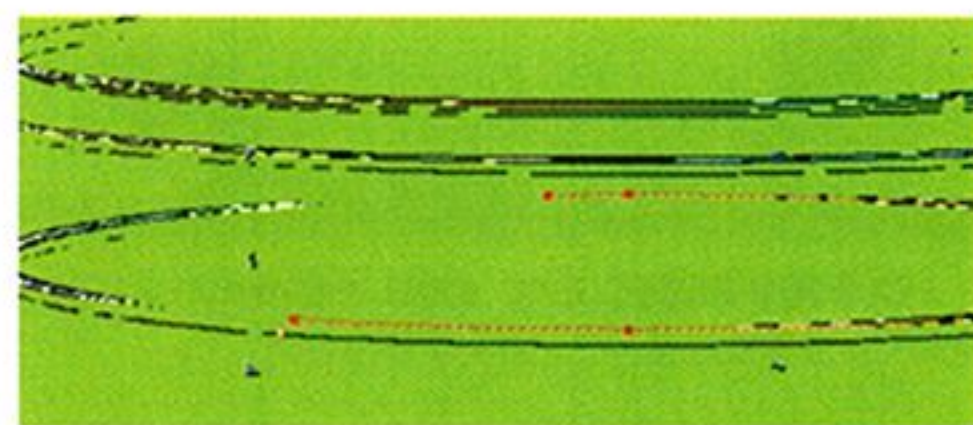



図93 パスを切断して2つに分ける。切断された一方は新しいオブジェクトとして最前面に作られるので注意

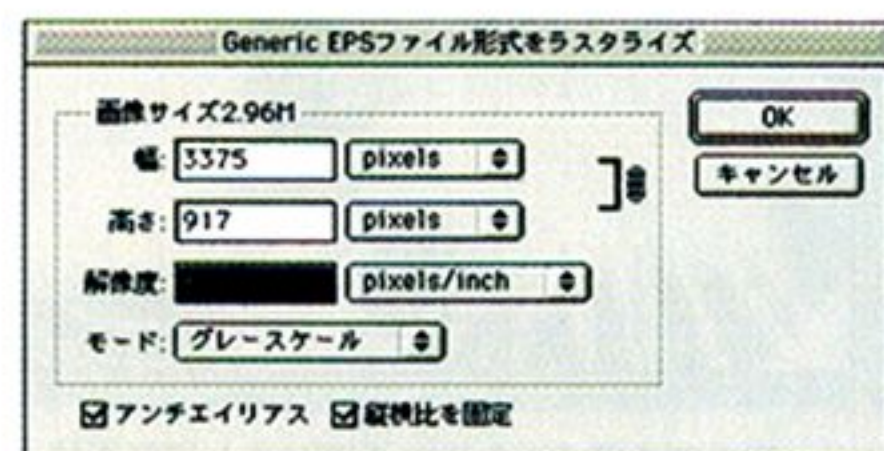



図94 Photoshopデータへの変換。アンチエイリアスはoffでいいかも

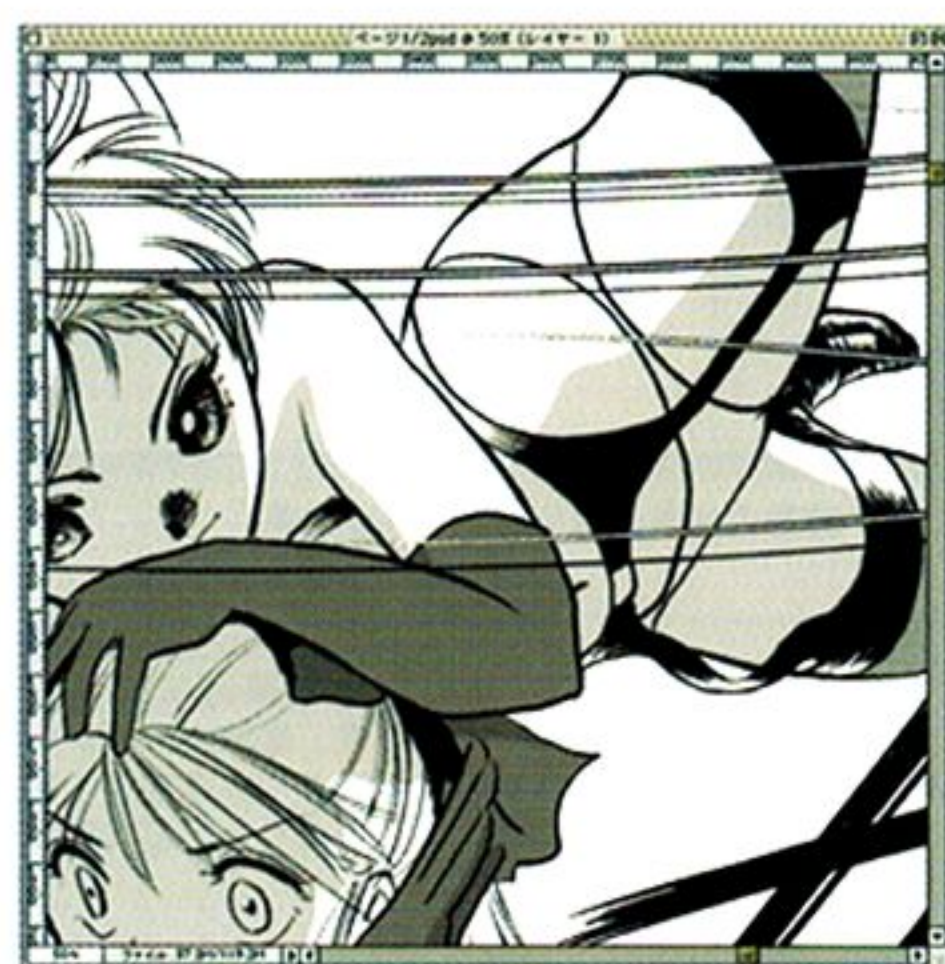



図95 流線を配置したところ。線の間が最初から白く抜けている

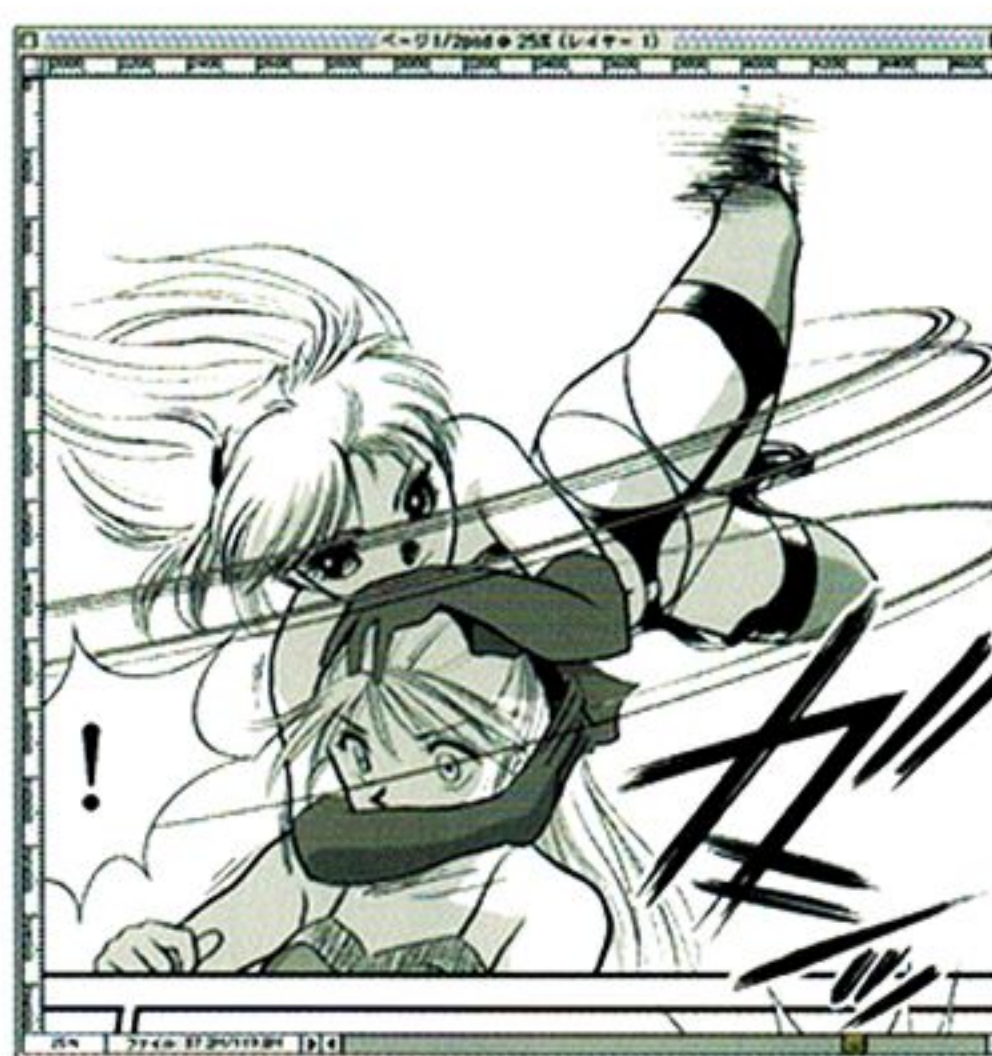



図96 キャラクターにあわせて傾ける

図97 回り込んでいるところはレイヤーマスクを追加して修正

図98 あとで配置を変更することも可能

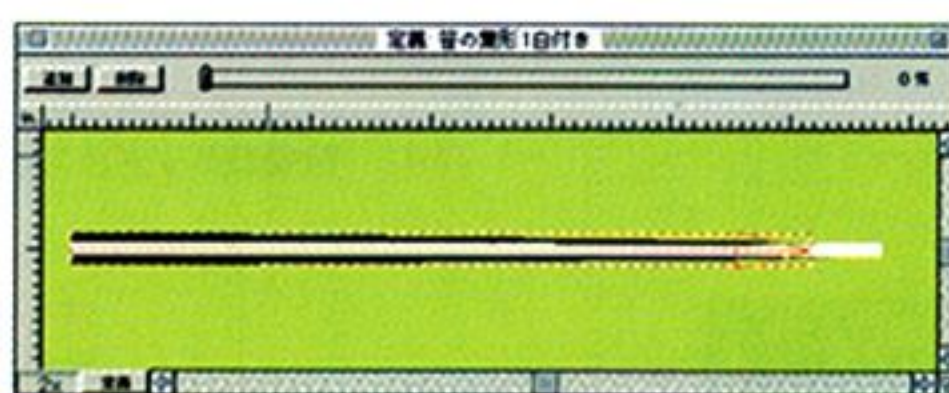



図99 スピード線にも白を入れる

図100 白抜き付きの集中線を配置

まず1コマ目の四角を選択し（図103），集中線レイヤーにレイヤーマスクを作成します（図104）。そうしておいて抜きの必要な部分の白を選択し（図105），コピーします。次にひとつ下に新規レイヤーを作成し「選択範囲内にペースト」を実行します（図106：単に白で塗り潰しでもいいかも）。これも1コマ目を選択してコマの形にカットします。集中線の先端部分をエアブラシで修正します（図107）。ほかのコマのスピード線・流線も同じようにして描いていきます。

トーン処理

先に別レイヤーに塗っておいたグレイ（図A）をアミトーン化します。服の色と影のレイヤーを2つに分けておいて本物のスクリーントーンみたいにアミをずらして重ねることも考えましたが，アナログ手法の模倣もほどほどにしておかないとなんだか本末転倒のような気がしたので，ひとつのレイヤーに統合して，一括してアミトーン化することにしました。

グレイレイヤーを「全選択＞コピー＞新規書類作成＞ペースト」で別ファイルにし，画像を統合したら「イメージ/モード/モノクロ2階調」で二値化します。設定はハーフトーンスクリーン（図B）で線数は55線，角度45度，網点形状は「円」です（図C）。

図Dは二値化されたアミを拡大したものです。データ入稿では出力機の線数によってはアンチエイリアスの部分にさらに細かいアミがかかってしまうため，モノクロ二値が推奨されていますが，二値のアミをそのままLaserShotで打ち出すと必ずどこかに横筋が入ってしまうので，グレイスケールに戻していったん50%に縮小し，再び原寸に戻してアミにアンチエイリアスをかけました（図E）。こうすればLaserShotの誤差拡散で打ち出したときに，綺麗な並びの網点になります（打ち出したものは一応，立派な二値ですし）。グレイスケールの網に変換したらShiftを押しながらドラッグ＆ドロップで原稿に戻し，「乗算」で合成します。これらの過程は一種の定型作業なのでアクション機能に登録しておけば作業は楽です。アミの線数ごとにアクション機能を用意しておいてもいいし，必要な設定ダイアログだけをその都度出すようにしてユーザーに入力させることも可能です。キャラのアミタイツ（?）部分はあらかじめこのキャラ用にIllustratorで作成しておいた六角形のパターン（図F）をPhotoshopデータに変換してパターン定義し，「塗り潰し/パターン」で塗り潰したものです。

もっといろんな種類のアミを使いたいなら，「PowerTone」という製品がおすすめです。Photoshopのプラグインという形で提供され，アミだけでなくいろんな種類のトーンがアナログトーンを貼り込む感覚で使用できます。この記事が読まれるころには「PowerTone2」が発売されてトーンの種

図103「枠線レイヤー」の一コマ目を選択

図101 線の先にかかったグラデーションの白部分が浮き上がっている

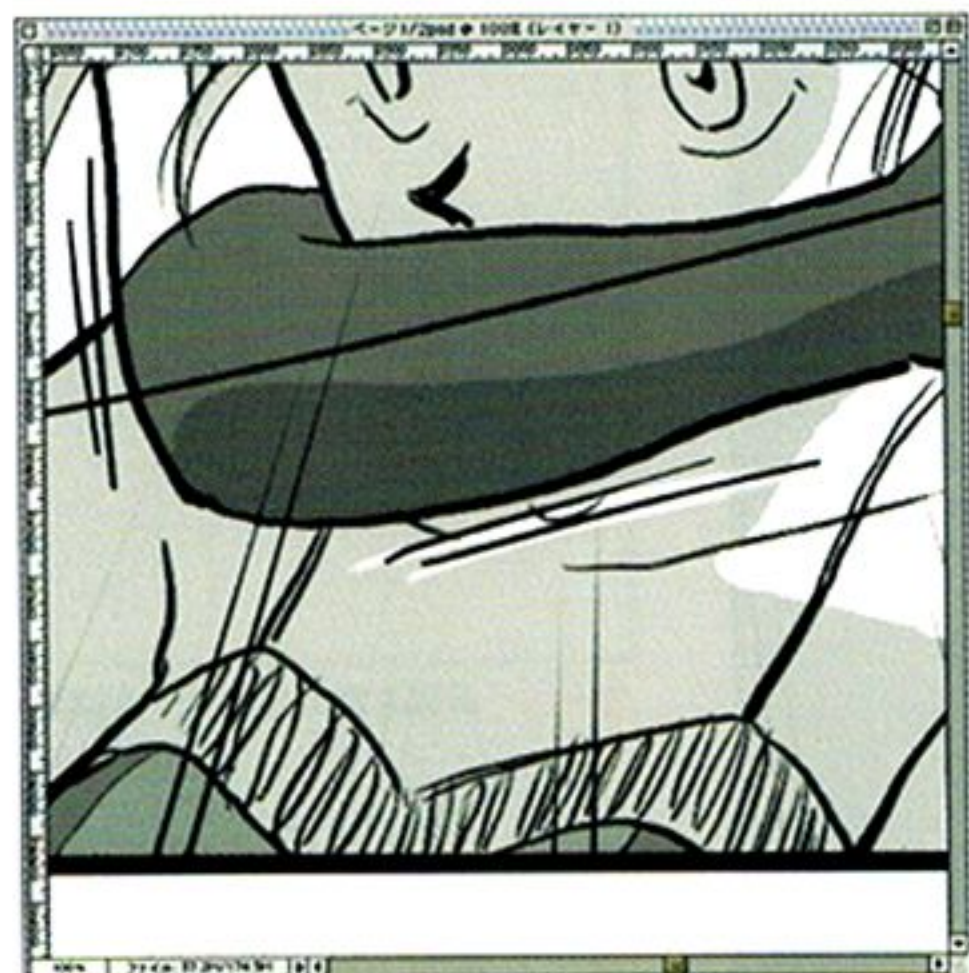



図102 乗算にすると線と線の間の白も消える

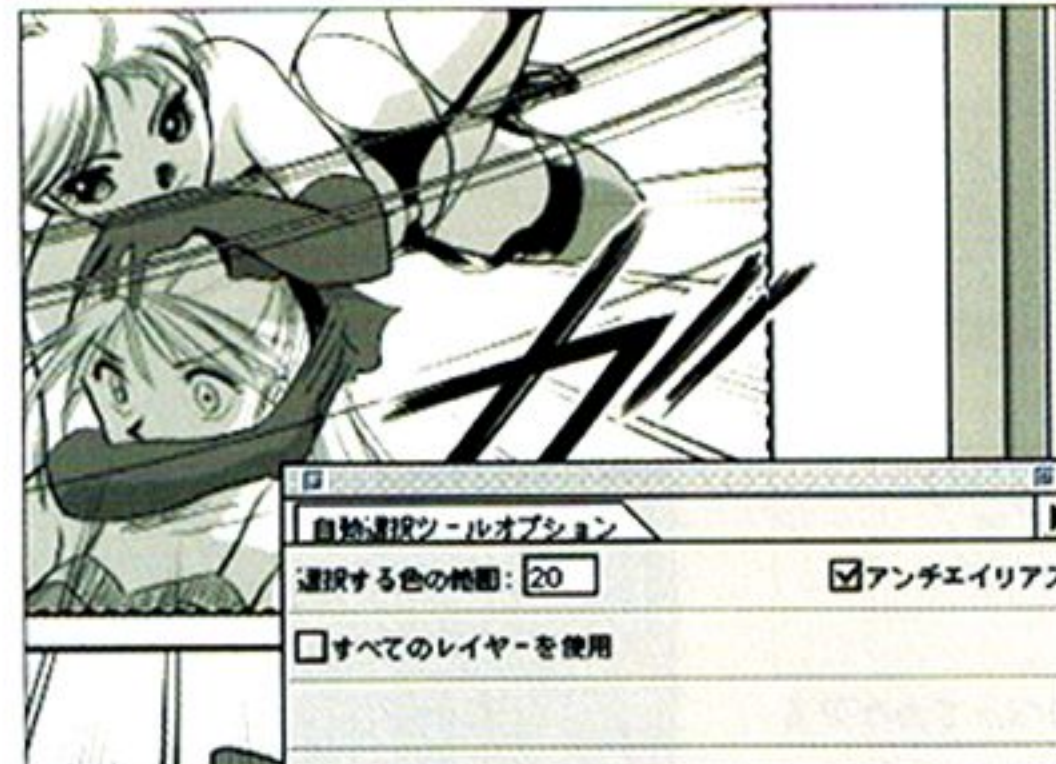



図106 別レイヤーを用意して白で塗り潰す

図104「集中線レイヤー」にレイヤーマスクを追加。コマの中に収める

図105「集中線レイヤー」の線と線の間の白を選択

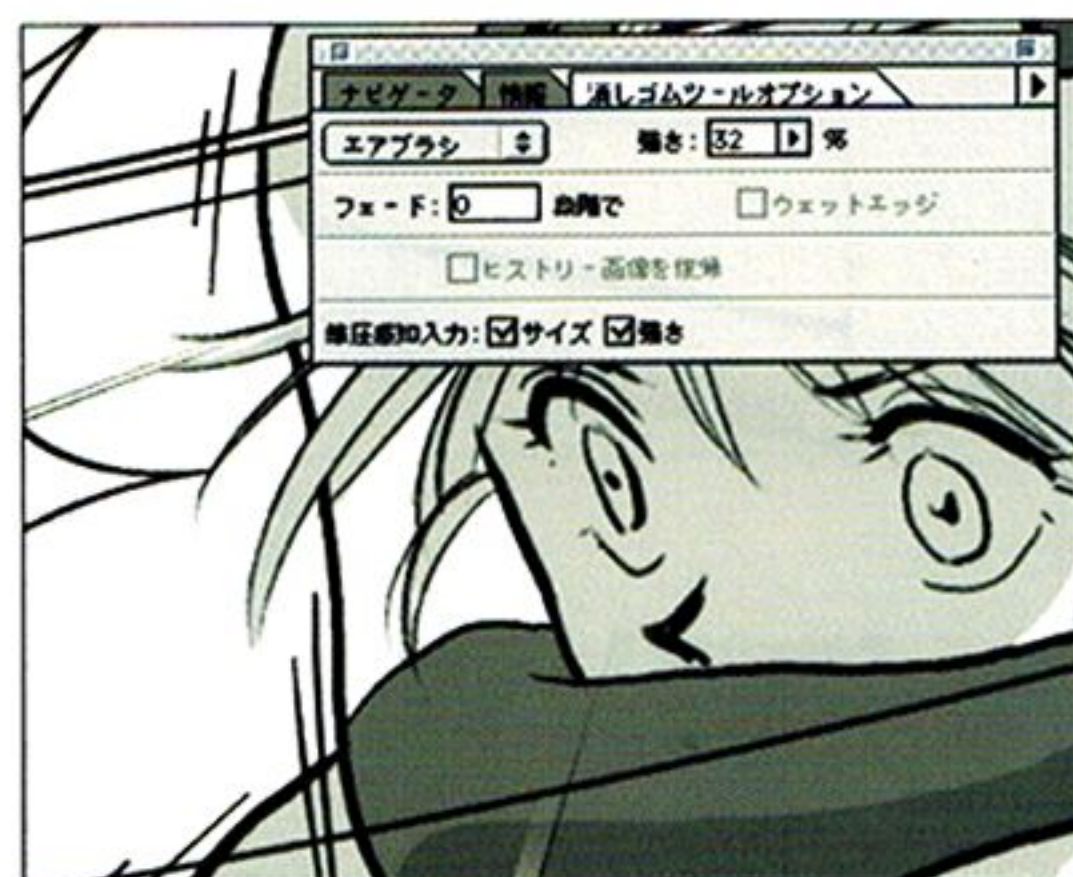
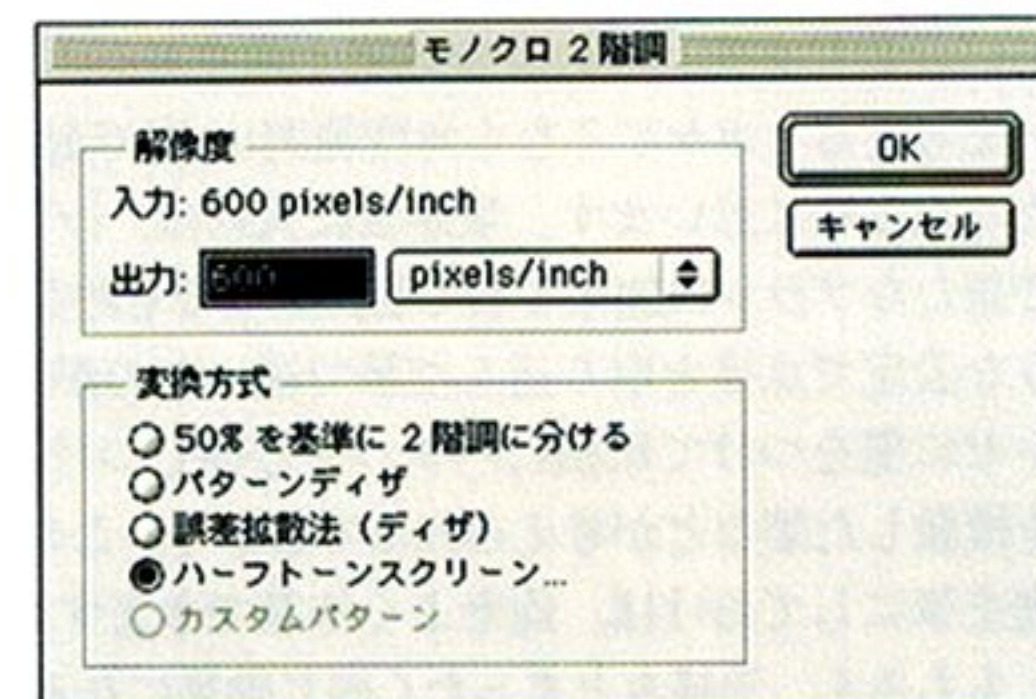
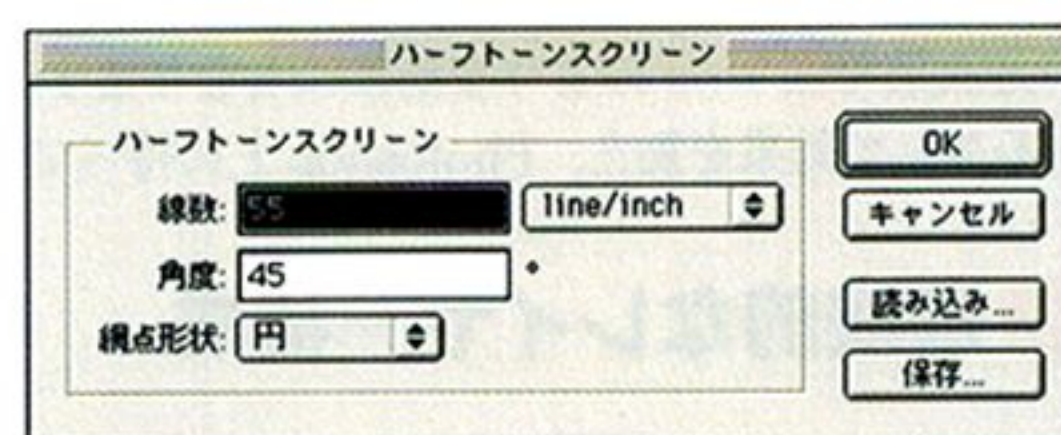
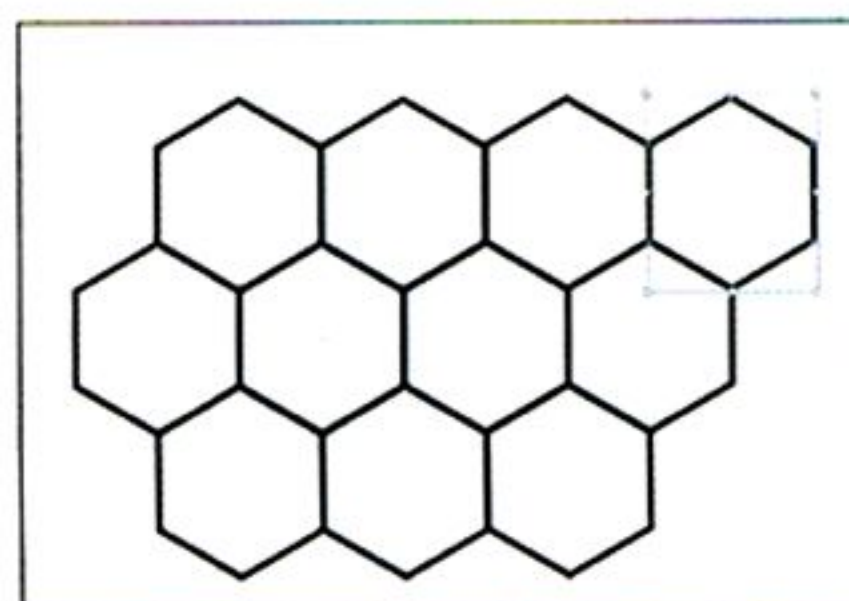
図B「ハーフトーンスクリーン」でモノクロ二値化

図C ハーフトーンスクリーンの設定

図107 白を「消しゴム/エアブラシ」で修正。レイヤーの合成方法を「ディザ合成」にしておくと手描き修正っぽい

図E レーザーショットで打ち出す場合には，アンチエイリアスのかかったアミの方が乱れがない

図A 先に塗っておいたグレイのレイヤー。地の色を白く塗ってある

類も格段に増えているはずです。

Painterのブラシ効果

このほか，マンガ表現のさまざまな効果において「黒山の人だかり」に使用したPainterの「取り込みブラシ」が役に立ちます。たとえば，図109のようなカケアミを描いておいて，矩形選択ツールで選択し，ブラシメニューの「ブラシの取り込み」を実行します。取り込んだときはその時点で有効になっているブラシの属性がそのまま引き継がれているので，図109のよう

図F Painterのペンでカケアミを描く

図D アミタイツ用のパターン

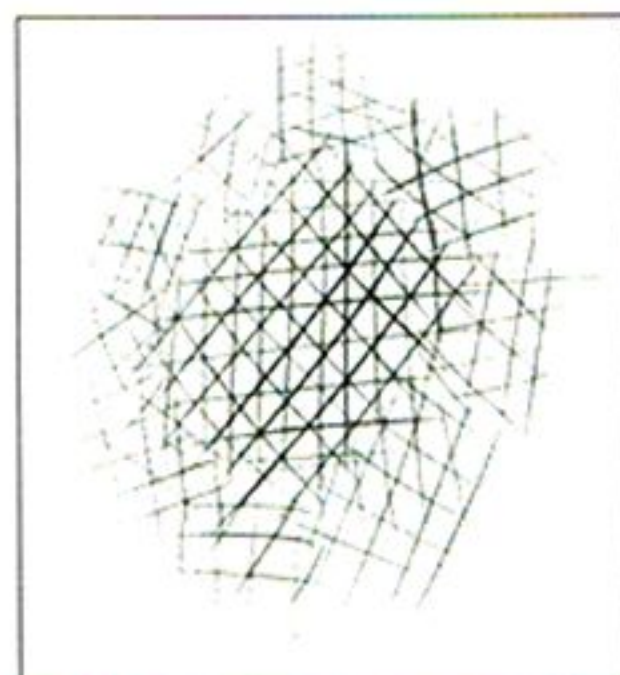



図108 Painterのペンでカケアミを描く

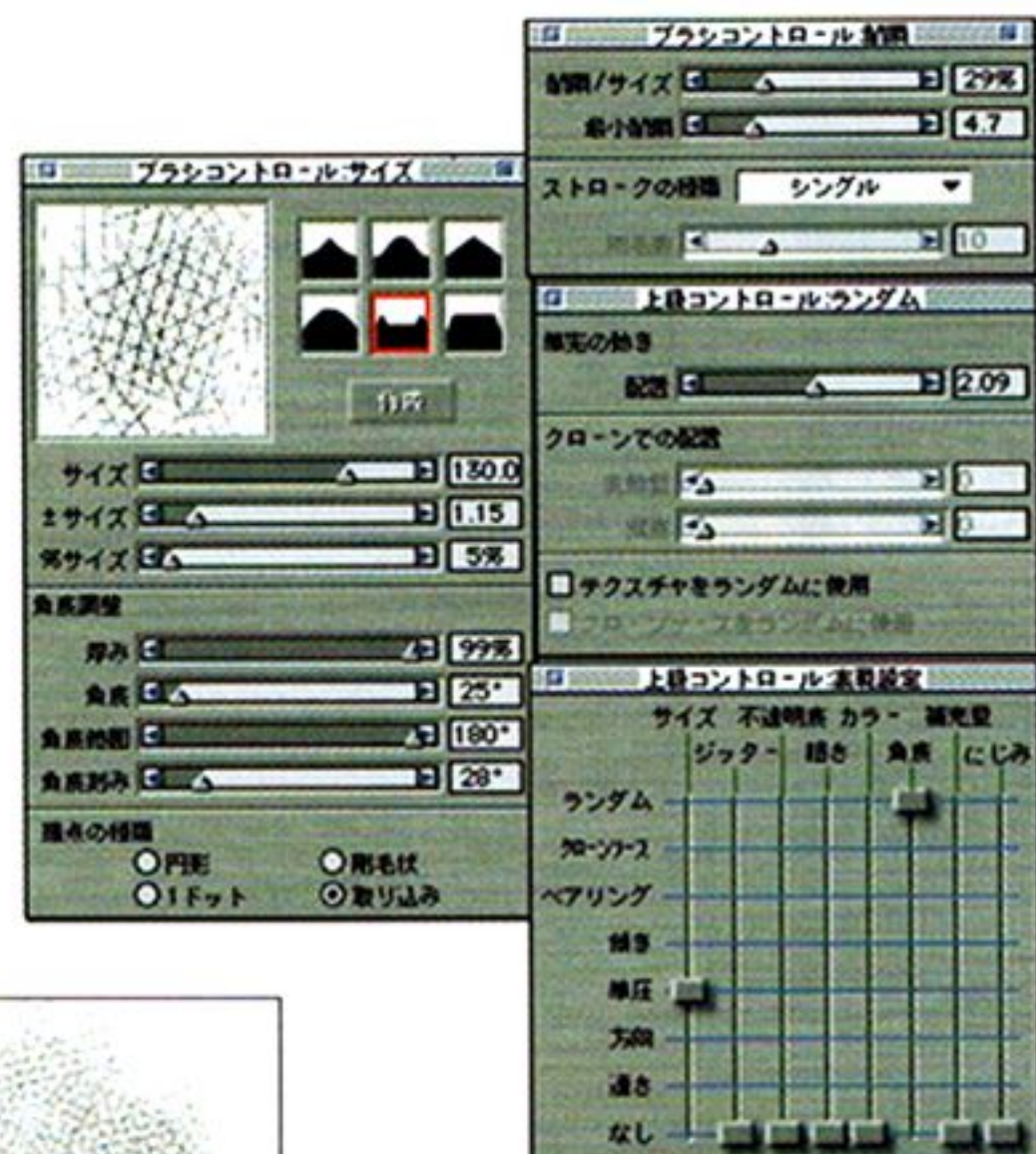



図109 カケアミをブラシとして取り込む。ランダムに回転しながら描画する設定に

図110 「カケアミ筆」の効果

図111 点描を取り込んだ「点描筆」の効果

図112 「ヘビ縄」失敗。これはこれで面白い効果

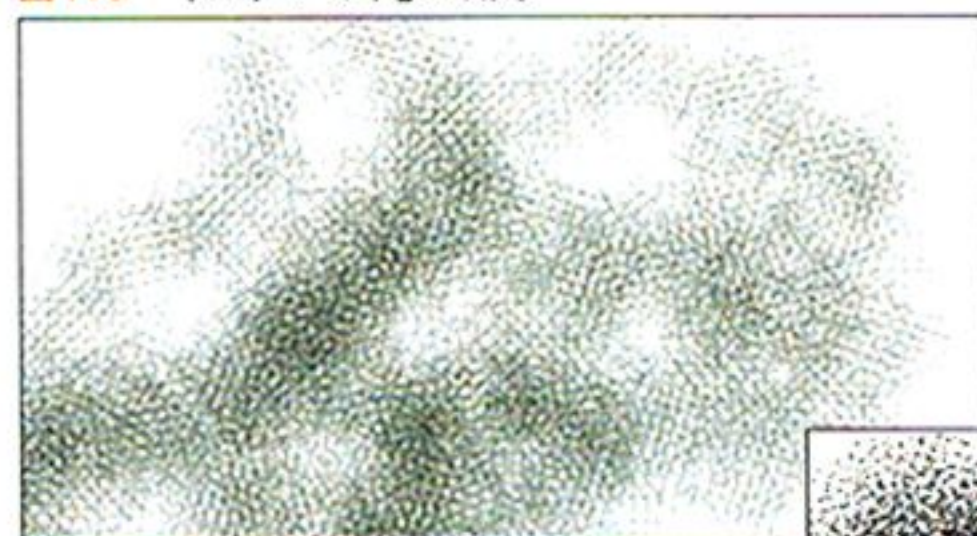
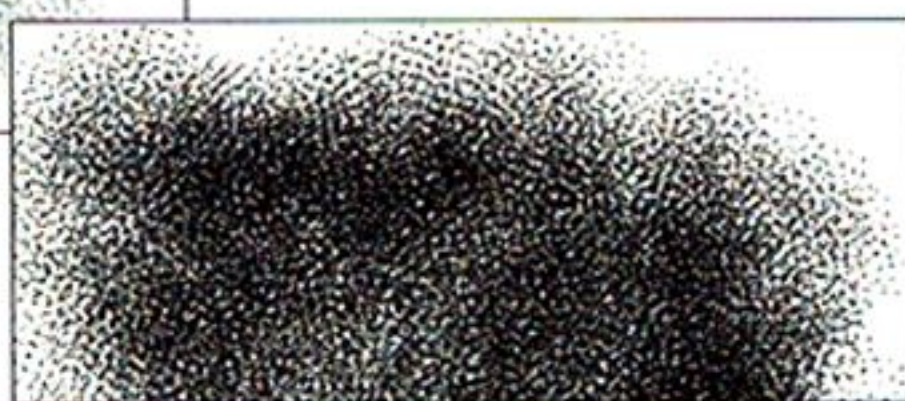



図113 点描筆にはこういう使い方もある

に間隔を適当にあけて，ランダムに回転しながら描画する設定にします。厚みが100%のままでは回転しないので厚みを99%にして角度範囲を180度に設定します。角度刻み値が小さいと処理が重くなるので，なるべく30度以上に設定します。この設定の筆で描くと図110のような効果になります。いま考えるとカケアミなら角度範囲90度で刻み値が15度のほうが賢い選択だったように思います。筆が気に入れば，「バリアント/バリアントの保存」で新たなブラシに加えておくと，いつでも呼び出すことができます。同じような設定で点描を取り込んだ筆で描いたのが図111です。このほかにもガーゼに墨をつけて原稿にパタパタと押しつけていく「パッティング」の手法を模倣した筆などが考えられるでしょう。このように時間のかかりそうな処理を筆にしておけば，効率よく作業できます。

もちろん，手描きとまったく同じ効果になるわけではありません。図112はいわゆる「ヘビナワ」を作ろうとして失敗したものですが，手描きのようで手描きの効果にもない味が出ていると思います。点描用に作った筆も，使い方次第で図113のような効果にもなります。この筆を図114のようにトレスして効果を加え，Photoshop上に持っていって合成します。

最終的なレイヤー構成

参考までに最終的なレイヤー構成を図115に示しておきます。いちばん下に「上・下」とあるのが2つに分けて描いたキャラクターのレイヤーです。もちろんこれはあくまでも参考であり，実践でこれだけのレイヤーをいちいち残しておくのはデータが大きくなりすぎて実用的ではありません。普通は統合できるレイヤーはどんどん統合していったほうがいいでしょう。ただし「通常」・「乗算」のように合成方法の異なるレイヤーをまとめるときは注意してください。

レーザープリンタの設定

キヤノンのLBP-750はMacではNetHawkというドライバでコントロールします。押さえておくべき設定は以下のとおり。

品質は「ファイン」。印刷は「LIPS」と「QuickDraw」の2種類があり，「QuickDraw」で印刷するとキレイではありますがアミに線が入ってしまうため（注：これもアミにかけるアンチエイリアスで回避できるレベルかもしれない），「LIPS」で印刷します。パターン表現は「細かく」で「精密ビットマップ」のチェックは外しておきます。「ビットマップスムージング」にチェックを入れていますが，本当のところ，効果に違いがあるのかどうかは自分でよくわかってません。いちばん大事なのは階調表現。これを網トーンではなく「誤差拡散」にすることで，スピード線の抜きをそれらしく表現します。

この設定を押さえておけば，生原稿と比べても遜色のないくらいの原稿が打ち出されます。

今回の作例は大体以上のような技法に尽きますが，おさらいとしてもうひとつ作例を示します（図116 イラスト）。特にプロレスのリングをなんのために3Dで作ったのかピンとこない人もいるでしょう。

まず，上の原稿で使用した3DのリングをShadeで開いて，あらかじめ用意しておいたキャラクターのプレビュー用画像をテンプレートとして読み込み，パースをあわせます（図117）。下からあおっているため，鉄柱などの縦パースが若干ついています。ここではパースはそのままにしていますが，漫画的二点透視法表現で縦のあおりパースを殺したい場合は，「あおり」補正をチェックしておきます（クイックレンダリングはoffにします）。そしてロープを変形させますが，このロープは円をパススイープしただけで，スイープに使用したパスそのものも保存してあるので，変形させたいロープを削

図116 イラスト

図119 Painterのペンでこのようなタッチを描いてブラシとして取り込む

図120 一筆でこのようなタッチを簡単につけられる

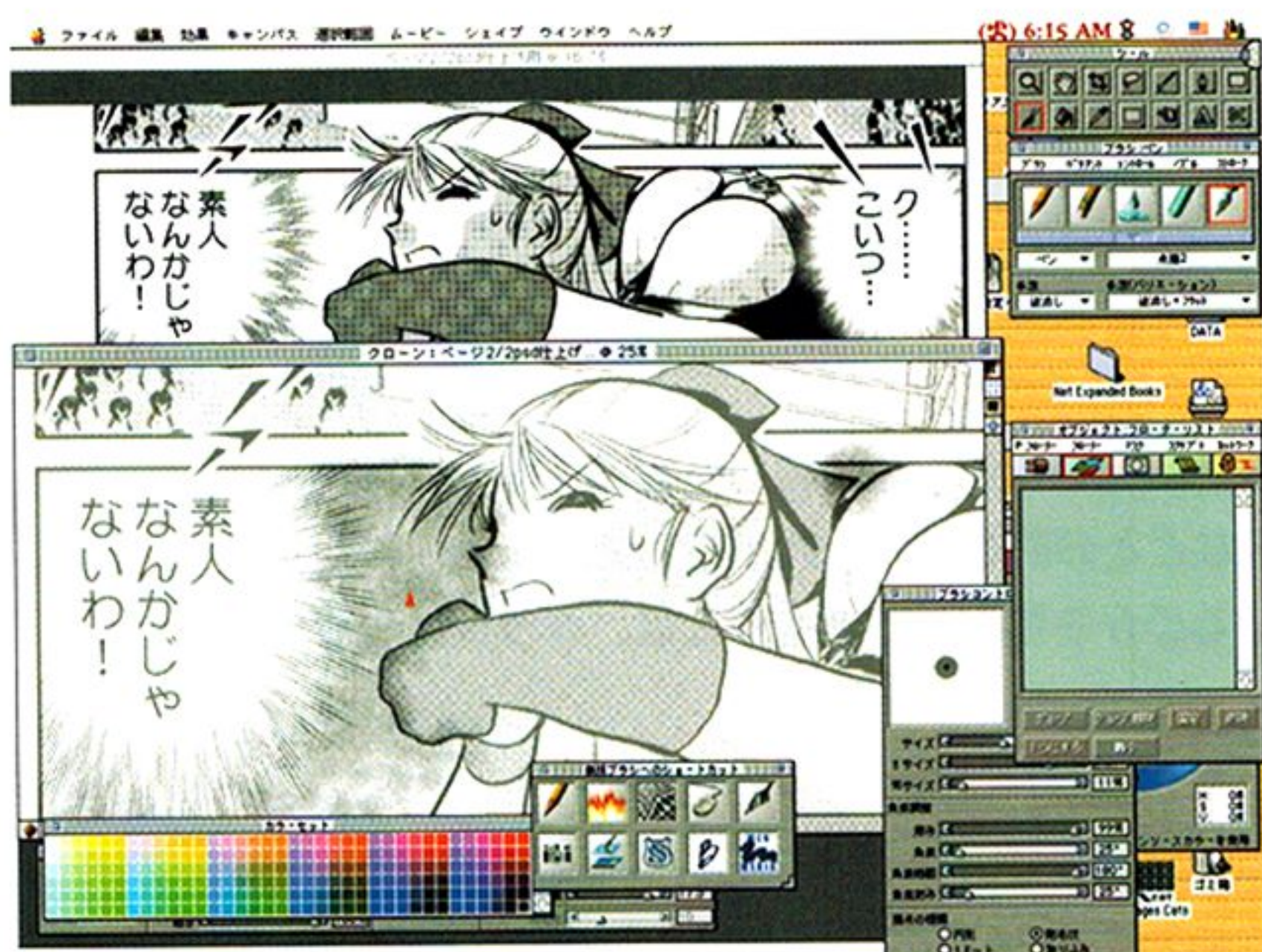



図114　トレスして効果を加える

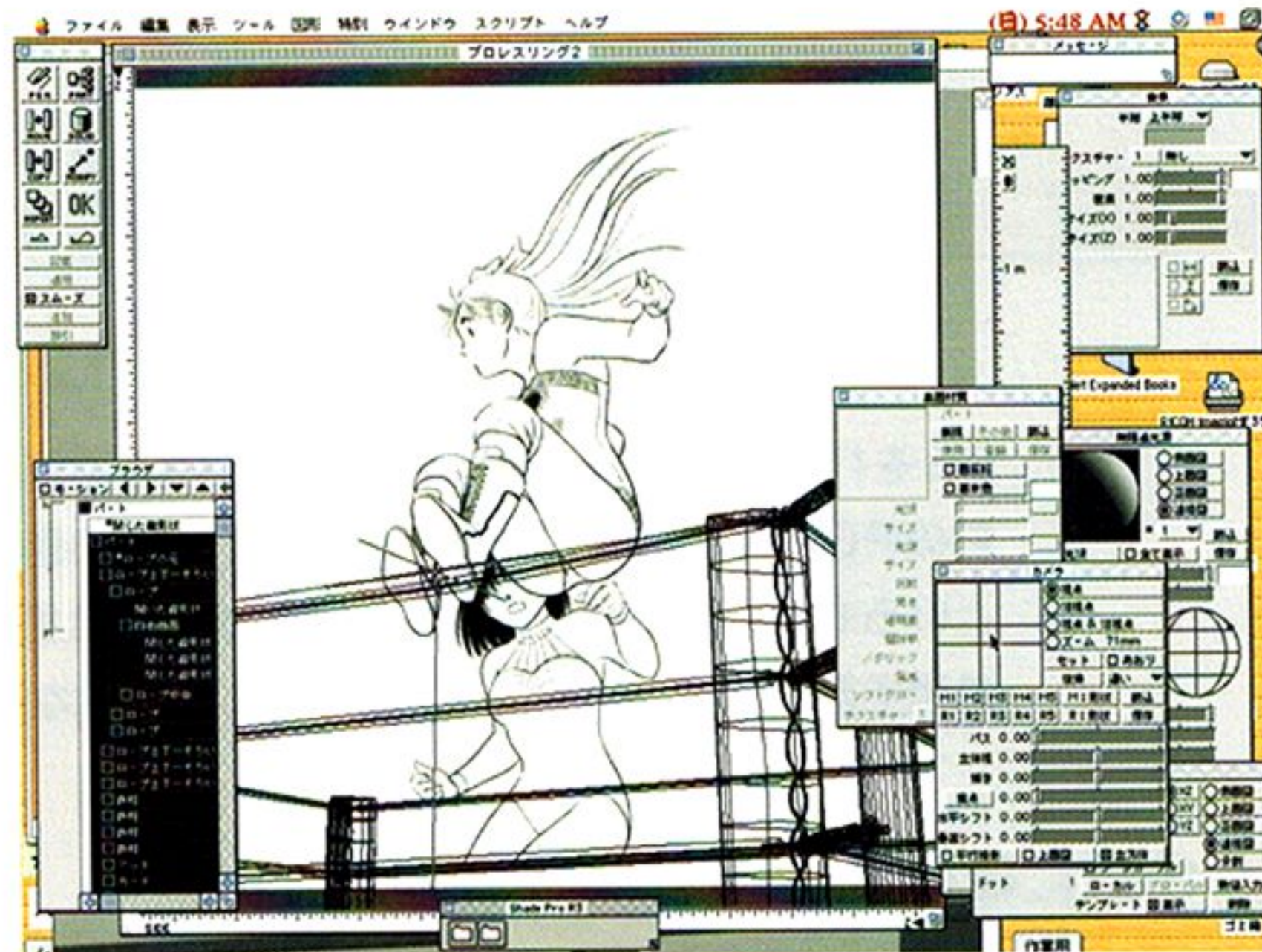



図117　キャラクターとリングのパースをあわせる

図115　最終的なレイヤー構成。これだとデータが大きくなりすぎるので，統合できるレイヤーは統合していく

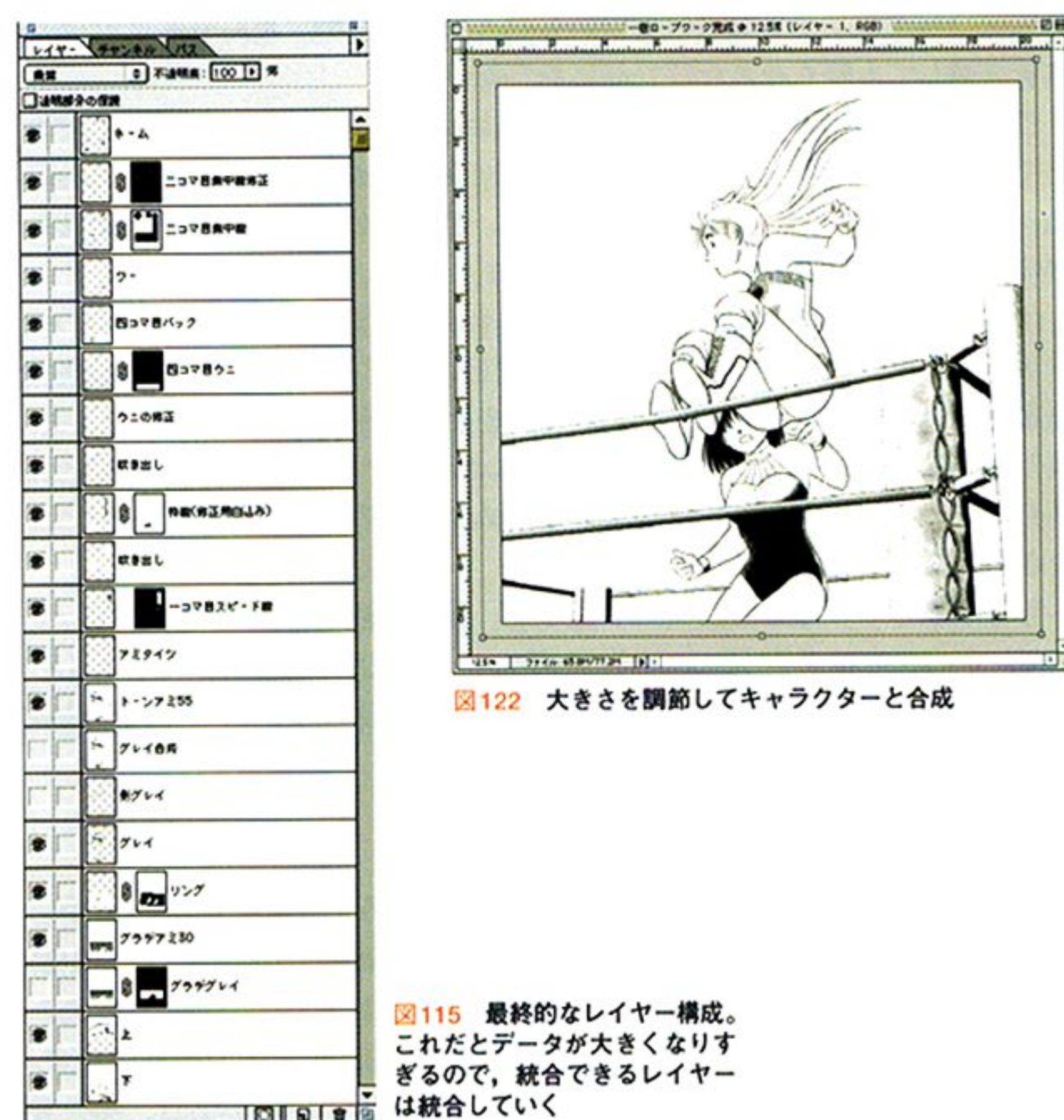



図122　大きさを調節してキャラクターと合成

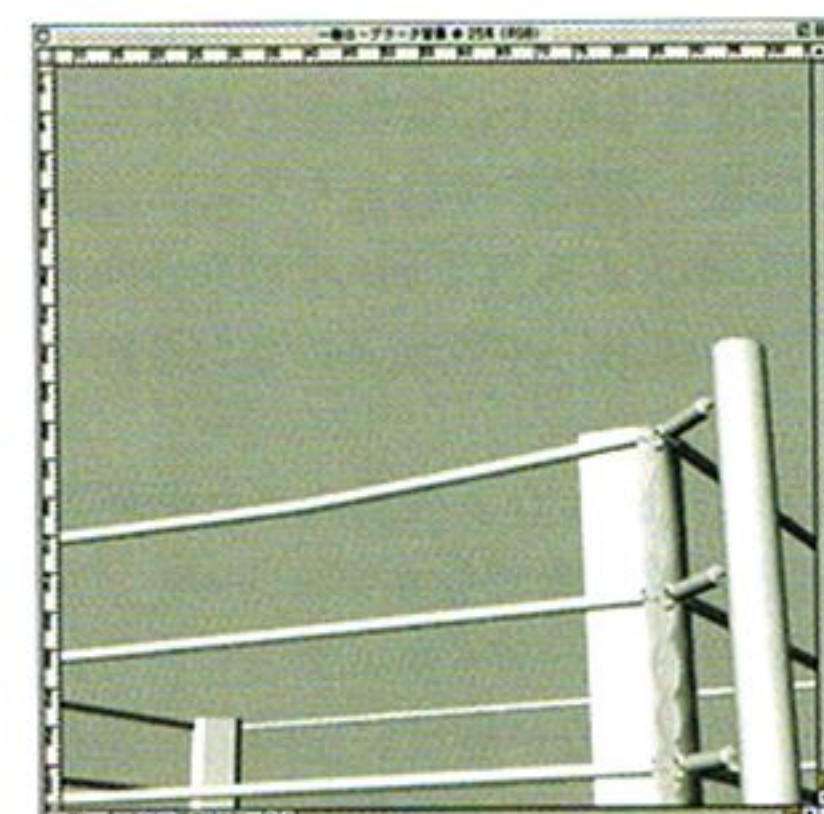



図118a　リングのレンダリング画像

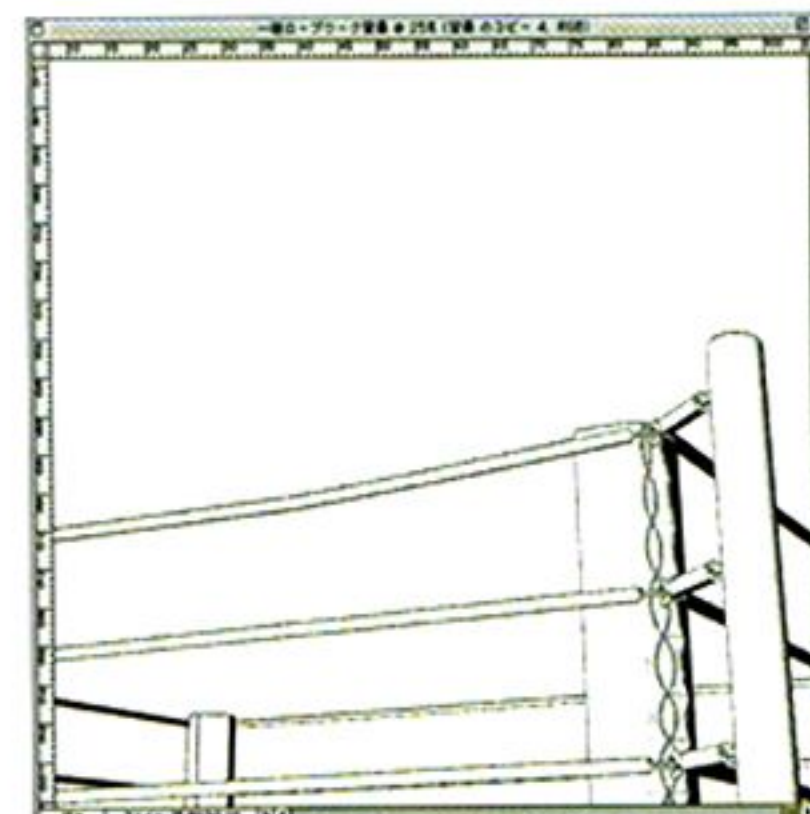



図118b　例によってアクション機能で主線抽出

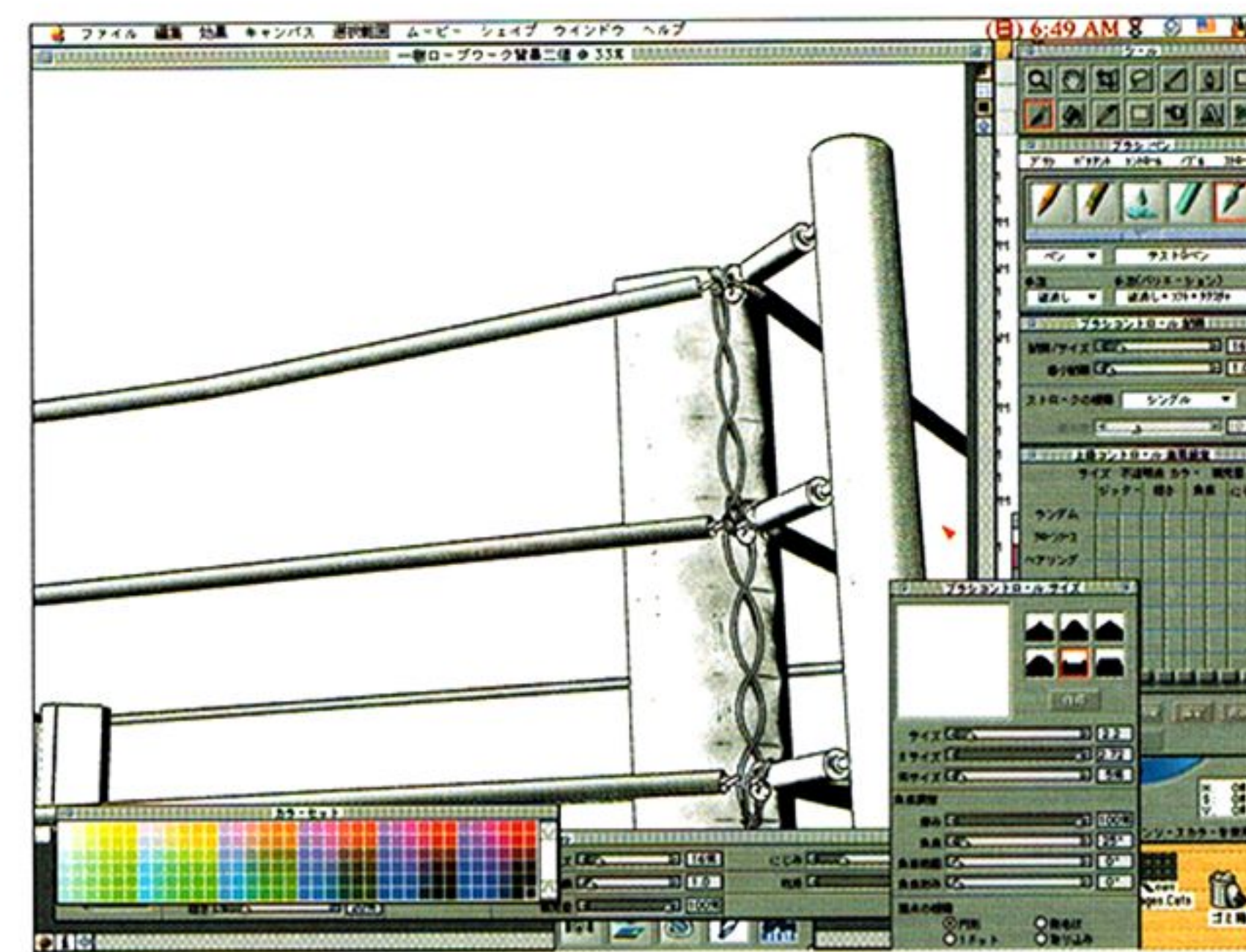



図121　効果を入れてリングの完成

除してスイープ用のパス(ベジェ曲線)を編集し，再びパススイープするだけで簡単に変形させることができます。ShadeのR2に付属していた「円の掃引」スクリプトがあれば，スイープする円の半径を入力するだけでロープのできあがりです。R3の現バージョンでは同様の機能がさらに強化されてプラグインツールとして追加されています。

次にレンダリングしたものをPhotoshop上で線画に変換します(図118a・118b)。これだけでは味気ないのでPainter上で効果を加えます。Painterのペンで図119のようなタッチを描き，「ブラシの取り込み」を実行して，筆の大きさを調節します。ロープ内部を自動選択ツールで選択して，取り込んだブラシを走らせると図120のようなタッチを簡単につけることができます。直線モードで描けば，あっという間にロープができあがりです。

さらに上で作ったカケアミブラシで効果を入れ，ペンで汚しを少し入れてヒモを描き加えれば完成です(図121)。Photoshop上でキャラと合成し，大きさと位置を調節します(図122)。あとはキャラにトーンを貼るなどして仕上げます。

最後に

以上，私なりのマンガ原稿の作成方法を紹介してきましたが，なにも新しい表現の可能なコンピュータを使ってまで，伝統的なマンガの描き方を踏襲する必要はないと思われるかもしれません。しかし，ひとつの選択肢として……，あるいは紙の原稿からデジタル技法へのつなぎとして，こういうやり方もあっていいと思います。

なお，今回細かく掘り下げられなかったところや，補足事項など書ききれなかった部分については以下のホームページにてフォローする予定です。
URL：http://www.amy.hi-ho.ne.jp/moriri/

色覚のお話

川原由唯　Kawahara Youi

CGを行ううえでもっとも基本となる「色」について根本的なところから見直してみよう。まずは色彩のRGB表現とはなにかを光の性質や視神経の働きなどとの関わりから問い直してみたい。「色」というものがどういうふうに定義され，認識されているのかを把握しておくことは，グラフィック処理を記述するうえで決して無益なことではない。

色はにほへど

「色」は定量的に測定できる普遍的な物理現象のように思えますが，本来はかなり主観に依存する性質を持ったもので，自分が他人と同じ色を見ているのかすら判断が怪しいものです（ATフィールドを破るかしない限り）。そのためか，色を問う学問の裾野は素人の想像する以上に広範なものになっており，美術・デザイン，物理・化学，情報・認知科学はもちろん，心理学や哲学の領域にさえ突っ込んでもまだ手にあまるものです。そのなかから今回は，生理学方面と，そこから導き出されるCIE xy表色系のさわりについて簡単に書いてみようかと思います。

最初にお断りしておきますが，この話は絵や2DCGとはあんまり関係ないかもしれません。それから僕自身は別に色の専門家でもカラーコーディネータの資格持ちでもなんでもありません（カラーコーディネータってこういうこと勉強すんのかな？）。そんでもって，誤解がない程度に意図的に話を簡略化したり端折ったりしている個所があるので，正確なことやもっと専門的に知りたい方はその筋の書物を漁ってみてくださいネ。

なぜRGBなの？

テレビのブラウン管は縦横に敷き詰めた赤緑青の蛍光体に電子ビームを当てて任意の点を光らせることで映像を映し出しているのはご承知のことと思います（並置加法混色）。この赤緑青のことを，Red，Green，Blueの頭文字を取って「RGB」ということもご存じですよね。

また，ペイントソフトで絵を描いた経験がある人なら，パソコンの画面も「ピクセル（画素）」という点の集まりでできていることは知っているでしょう。ピクセルも，テレビと同様，RGBの発光の強さを調節することで，さまざまな色を表現しています。パソコンで扱う画像データ（ラスターイメージ）は，基本的にはこのピクセルの並びとそれぞれのRGBの値を用途にあわせたフォーマットで格納したものです。コンピュータによる画像処理アルゴリズムも，究極的にはRGB値とその並びを解析，操作することと説明できるでしょう。

さて，ではなぜテレビやコンピュータはRGBで色を表現しているのでしょう。「赤緑青が光の色の三原色だから」だって？　正解です。ふふふ。そりゃそうですよね。RGBは光の三原色といわれていますよね。三原色を混ぜあわせれば，ほぼあらゆる色調を出すことができます。それが原色の原色たる所以です。だからパソコンやテレビはRGBの3色を用いて色を描き出すのです。

それじゃ，なんで光の三原色は赤，緑，青なんでしょう。物理現象だから？　この世界の構造が最初からそうなってるから？　神様が決めたこと？　本当かな。

光の色とは？

思い出話から。

小学生か中学生の頃，LEDをプラモデルに組み込んで遊んでいたことがありました。あるとき，緑に光らせたいパーツがあったのに，悲しいことに手持ちのストックには黄色いLEDしかなかったんですね。買いに行くお金もない……。でも，貧乏人は創意と工夫で生きていくのだというわけで，青のセロファンを被せて緑に光らせようと考えました。ところが，何枚青のセロファンを重ねても出てくる光が黄色にしか見えない。不思議……だって豆電球ならセロファン通せば色づいて見えるじゃない。なんで黄色のLEDは青を通しても暗くなるだけで緑にならないの？

当時は不思議で不思議で，自分の目がおかしいんだと思って納得してたんですけど，いまになって考えれば，LEDの光が黄色の単色光（Monochromatic Light）だったからなんですね。要するに，混色で作られた黄色ではなくて，黄色成分だけの光のことです。

ほかに単色光の効果が簡単に見られるものといえば，トンネルの中のナトリウムランプがあります。あのオレンジ色は混じりっ気のないほぼ単波長の光なので，青の（分光分布を持つ）物体をナトリウムランプの下で見ると，ほとんど真っ黒に見えます。夕暮れの光もオレンジ色ですが，オレンジ色以外の波長の光がたくさん含まれているため，青いものを見ても真っ黒にはならず，ちゃんと青く見えます。

さて本題です。光は電磁波の一種です。粒子と波の性質を同時に持っていて，その速度は真空中で秒速約30万km……なんて話は今回はほとんど関係ないので物理の教科書に任せておきます。とりあえずここでは，波長と色の関係についておさらいしておきましょう。

光の波長

人間の目が捉えることができるのは，電磁波のうち，波長が400nmから730nm（広く取る説では，380nmから780nm）程度のものです。波長が長い側（730nm）を赤，短い側（400nm）を青紫，これらの中間は，お馴染みの虹の色「赤橙黄緑青藍紫」のような連続した色の並びとして感知します。なお，紫〜赤紫に相当する波長の光は存在しません。紫〜赤紫は，赤と青の混合によって見える色です。

波長というものは物理的には連続的に変化できますから，色の変化も波長に即した連続したものになりそうなものです。しかし実は滑らかな連続になっているとは限りません。人間の目では，異なった複数の特性を持つセ

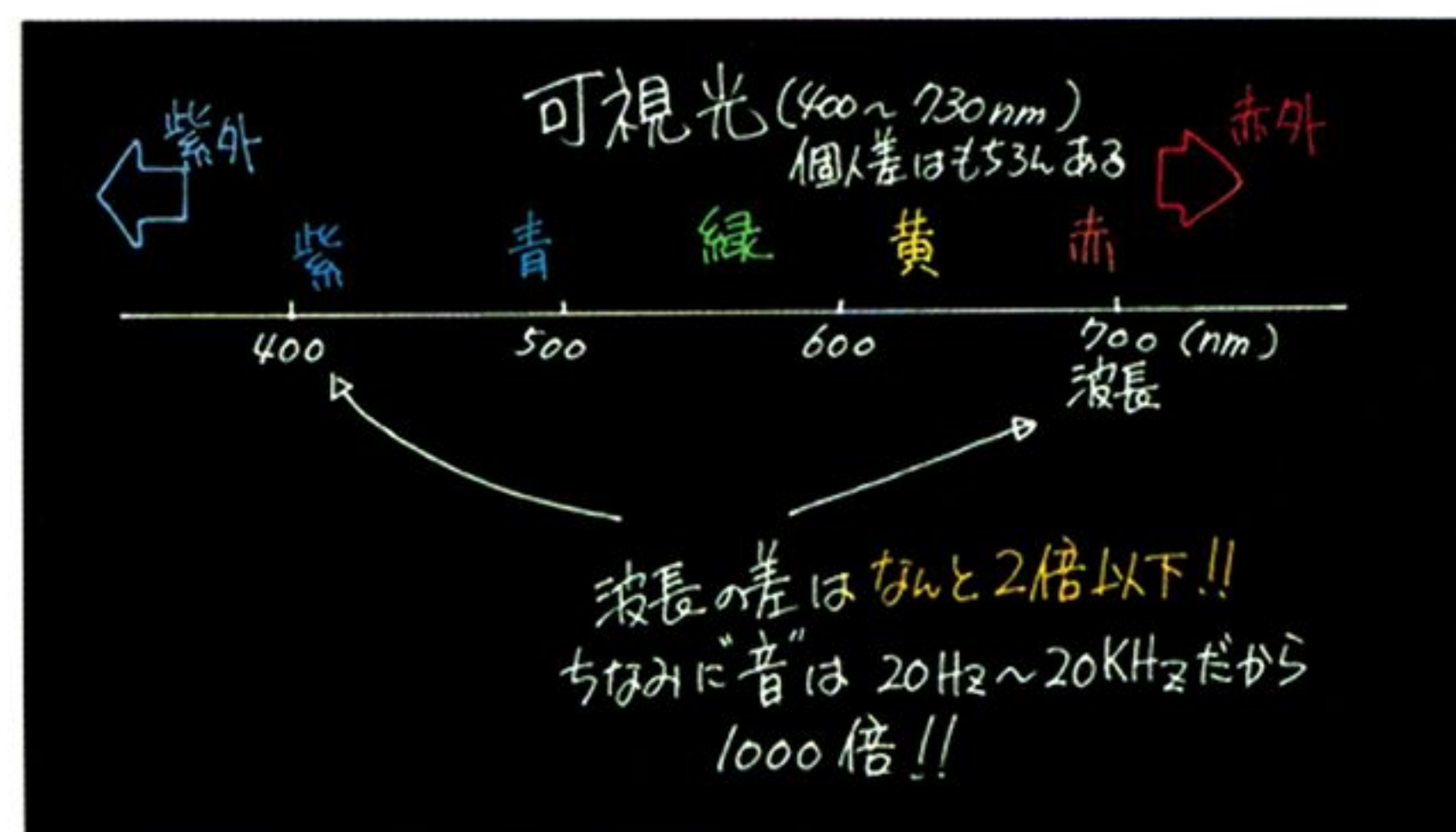
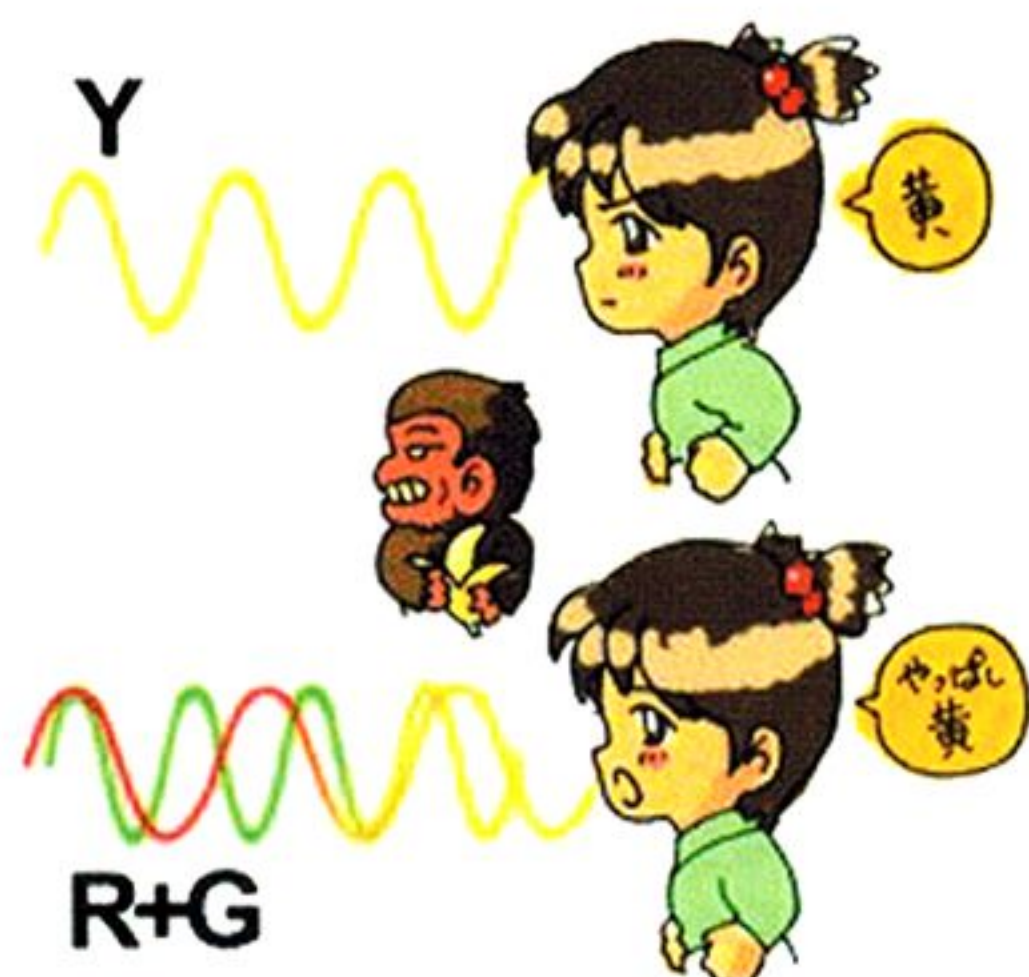



図1　光の波長

ンサ（視細胞）が協調して同時に働くため，目（網膜）は波長の変化に対して必ずしもスムーズに色を変化させてくれるものではないのです。

さて，いまの波長の話で出てくるのは，混じりっけのない単波長の光，いわゆるスペクトルに分光したときの色のことですが，それとは別に，異なった波長の光を合成することでも色は作ることができます。もちろんRGBの原色以外の単波長の光を混ぜ合わせても色は作れます。実際，RGB以外の3色を原色として定義しても，たいていの色は合成できます（ある1色がほかの2色で合成できる場合を除く）。どれだけの色を合成できるかは，選んだ3色の周波数や組み合わせによって変わってきます。いちばん多くの色を合成できる組み合わせがRGBだと思っておいてください。

異なる波長の光の合成の色と，単波長の純粋な色とを，（一部例外を除いて）人間の目は区別できません。たとえば，580nmの波長の光は黄色です。でも赤（たとえば630nm）と緑（たとえば510nm）の光を混ぜ合わせても黄色になるのは三原色の原理のとおりです。単波長の黄色と，緑と赤の合成された黄色は人間の眼には区別することができないのです。

以上のようなことを光と色の基本知識として覚えておいてください。

網膜を見てみよう

ここで，眼の構造を見てみましょう。

眼の構造と錐体桿体分布

眼球の内側には，眼に入った映像を捉えるための網膜と呼ばれる器官があります。デジカメなんかに使われるCCDと似たようなもので，ここに光を感じる細胞がびっしり並んでいることは予想できますね。この光感応細胞は大きく2種類に分類することができ，ひとつは明るさ（光の強さ）のみに反応する「桿体（Rod）」，もうひとつは色（ある決まった波長の光）に反応する「錐体（Cone）」となっています。錐体は，赤，緑，青のそれぞれの波長に最大に反応するピークを持つものが3種類あります。

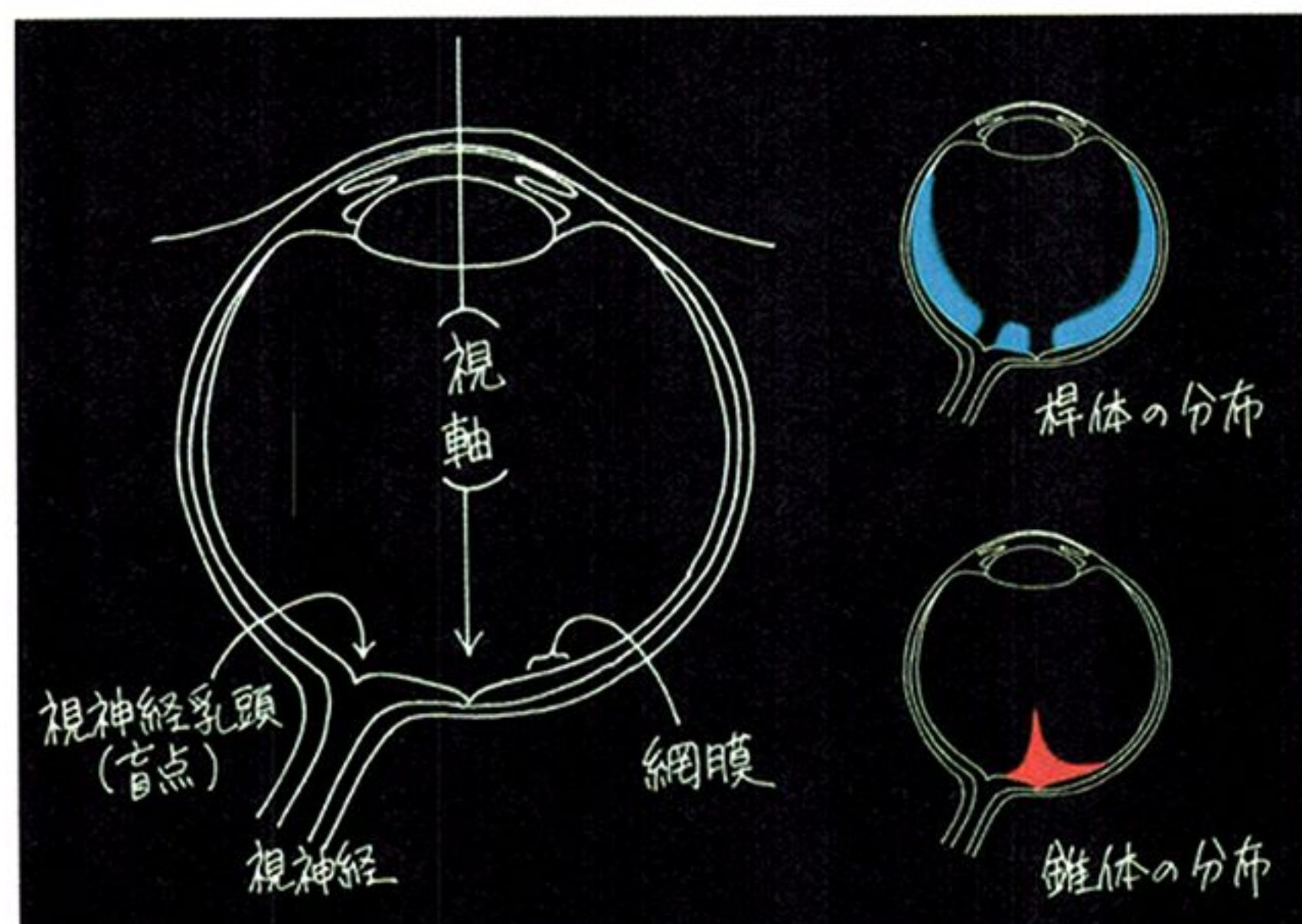



図2　眼の構造と錐体桿体分布

桿体細胞と錐体細胞

この「最大に反応する」ピークがあって，ほかの波長の光にも少なからず反応するというところに色感覚の複雑さのミソがあるんですが，それはおいおいわかると思います。

桿体は色を選り好みしない分，微弱な光を感知することができます。暗い場所でものを見るときに働くのは主に桿体のほうです。一方，錐体のほうは光量が少ないと役に立たないのですが，赤緑青の波長を見分けて色鮮やかな世界を脳に伝えます。

もうわかりましたね。なぜ光の三原色はRGBなのかはこの錐体がRGBに対応しているからなんです。人間の眼がRGBを特別に扱うから，RGBが三原色になったといえます。三原色を決めたのは人間の眼の構造なのです。ミツバチは紫外線を見られる（正確には視覚が人間よりも紫外線寄りにシフトしている）というのは有名な話ですが，もしも人間の眼に，赤外線か紫外線に反応する「第四の錐体」があったら，光は四原色になったのかもしれません。四原色で構築された世界……想像できる人はすんごいアーティストかもしれませんね。

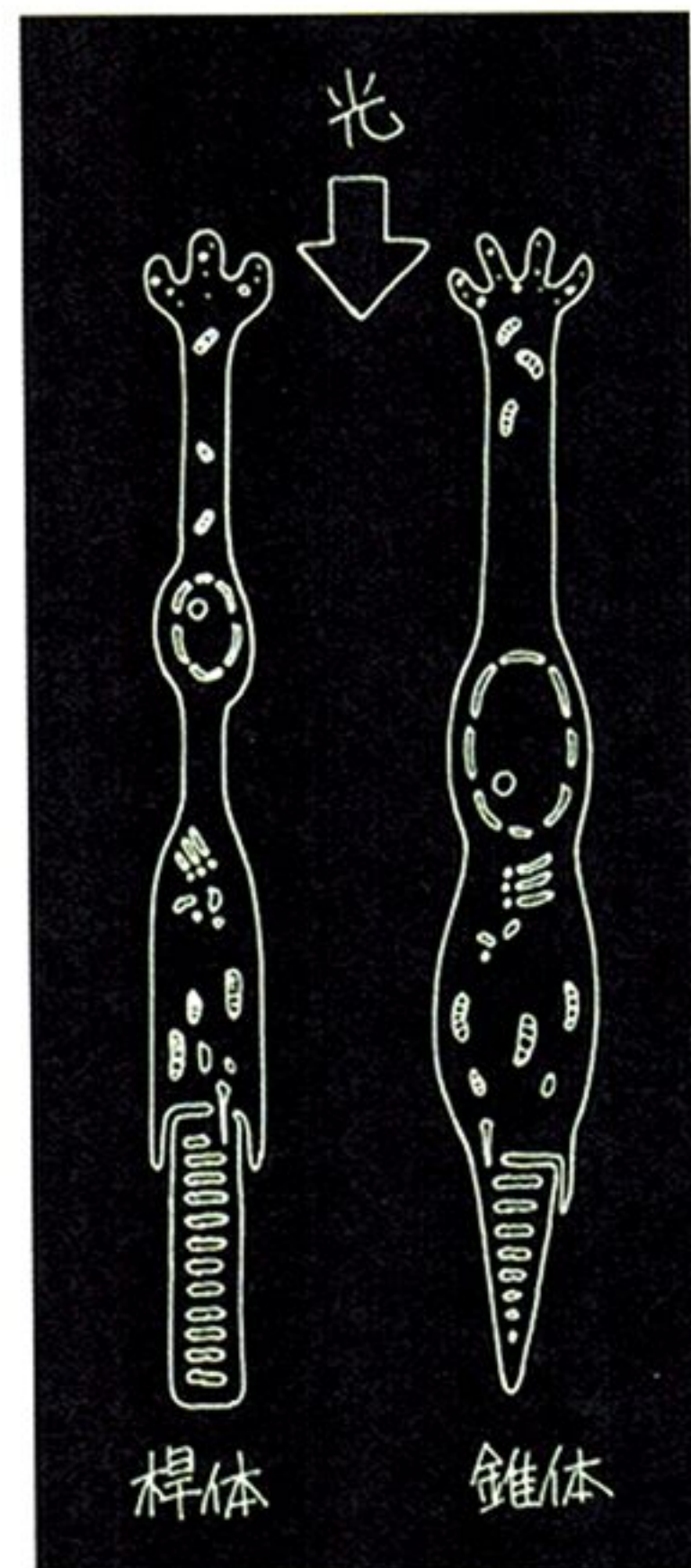



図3　桿体細胞と錐体細胞

視神経系

面白いのは，人間の網膜の場合，錐体や桿体が位置するレイヤーよりも上のほうに神経束が位置していることですね。つまり光は，それら神経の網の中を通り抜けてから光感知細胞に到達するということ。この構造は生物によって異なっていて，タコやイカなど，人間とは逆に伝達神経系のほうが下にある生き物もいるそうです

また，錐体，桿体などの視細胞から脳に向かって送られた映像情報は，脳に到達する前でも，さまざまな処理が行われています。つまり，眼や視神経系自身が，それ独自で情報処理を行っているということです。これは眼という器官が，発生学的に脳とかなり近い性質を持ったものであることのひとつの表れといわれています。また，この過程を経るために，人間の色の感

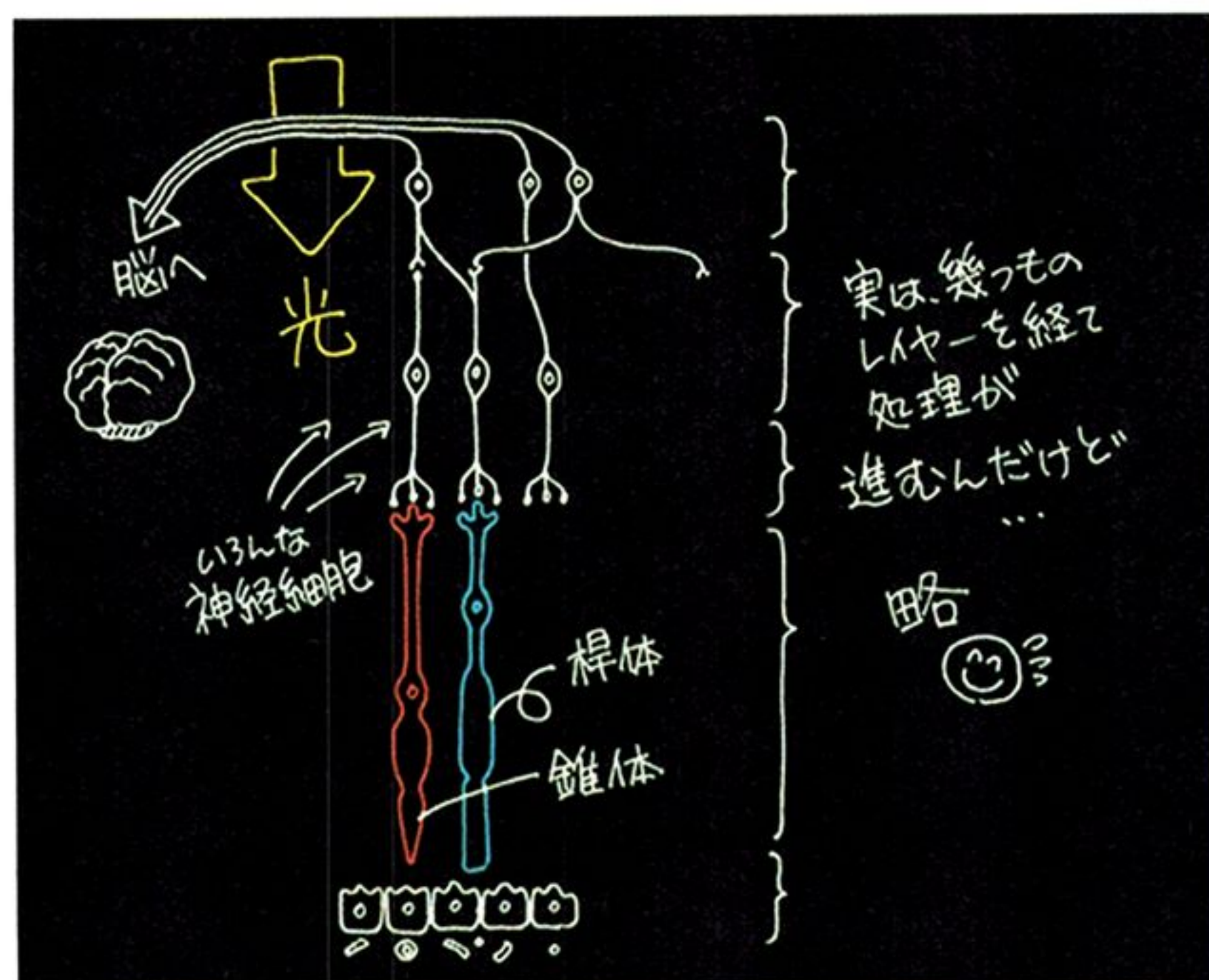



図4　視神経系

覚特性が，単純にRGBに還元できるものではないということにも注意が必要です（テレビ放送の技術，というか画像圧縮関連技術で出てくる“YC分離”は，このあたりの特性を利用して，重要な情報とあんまし重要じゃない情報に重みをつけて，電波という限られた資源を有効活用している）。

ちなみに錐体は網膜上の視軸（視線）の中心に多く分布し，中心から外れるほど少なくなっていきます。

逆に桿体は視軸の中心には存在していませんが，視野のちょっと脇のあたりにもっとも多く分布し，やはり中心から外れるに従って少なくなります。しかし錐体よりはうんと遠くまで広がっています（ただし「盲点」には視細胞はない）。

つまり，色は視野の中央でしか感じることができません。視線から大きく外れたところでは明暗のみ感じて色の判断はできないのです。また，錐体がひしめきあう視野の中央は，色は見えても微弱な光には反応しにくいため，暗いところでは斜交いにものを見たほうがよく見えます。たとえば夜空の暗い星を見ようとしたときに，対象となる星を見つめず，若干視線をそらしたほうがよく見えるということは，星を観測する方なら経験的に知っているんじゃないでしょうか。これは網膜上の桿体の分布状態によるものです。

錐体の分布について実感することはほとんどないと思いますが，僕はたまたまこんな経験をしてからそのことを確認しました。興味のある方は試してみてはいかがでしょう。

夜，横断歩道で信号待ちをしているときに，車道の緑信号を徐々に視線をそらして見てみましょう（夜のほうが，ほかの光の影響を受けないのでわかりやすいです）。そらしていくにつれて，緑だか黄色だかわからなくなるところがあるはずです。ただし，赤信号についてはかなり目をそらしてもわかります。赤に反応する錐体がほかの錐体に比較して端のほうまで分布するためのようです。

青い夜は本当に青いか

絵画やイラストでは，夜の風景を表現するときに青系の色を多く用います。普通に考えると，暗くて色が見えなくなる（暗闇では色を感じない桿体でものを見るようになる）んだから，無彩色のグレーや黒を多用しそうなものですが，グレーや黒を使うよりも藍色などの寒色系の色で描いたほうがより自然な感じになります。実際に夕暮れの風景は，夕焼けの茜や残照が収まっていくにつれて青く深く沈んでいくように感じられると思います。

このあたりは，プルキニエ現象（Purkinje-phenomenon/-shift）を考えると理解できます。これは，錐体と桿体の最大感度のピークが異なっていて，桿体のほうが錐体よりも短波長側，すなわち青のほうに寄っていることから起こる現象です。薄暗くなると赤系の色が見えにくくなり，青系の色は桿体によってキャッチされやすくなります。桿体自体が青い色を認識しているわけではないのでしょうが，相対的に青系に反射する物体のほうが明るく見えてくるわけです。というわけで，闇夜を描くときに青く描くのも，生理学的に正しいことになります。

CIE色度表はなぜあんな形なの？

CIEのxy色度表というのをご存じでしょうか（図5）。これは国際照明委員会（Commission Internationale de l'Eclairage = CIE）という組織が色を分類する尺度として1930年代に作成した規格の一種です。

一度くらい見たことはあると思うのですが，色の並びがなんでこんな変てこな形で表現されているのか，その理由と意味を理解されている方は意外と少ないと思います。よく見ると，座標軸とおぼしきものがRGBでもHSVでもましてやCMYKでもなく，なんとXYだったり，赤から青までは曲線なのに紫の線が直線だったりと，PhotoshopのHSVスライダーやPainterの丸い色パレットに慣れた我々には不可解な構造をしているのに気づきます。

マッハバンドの話　Column

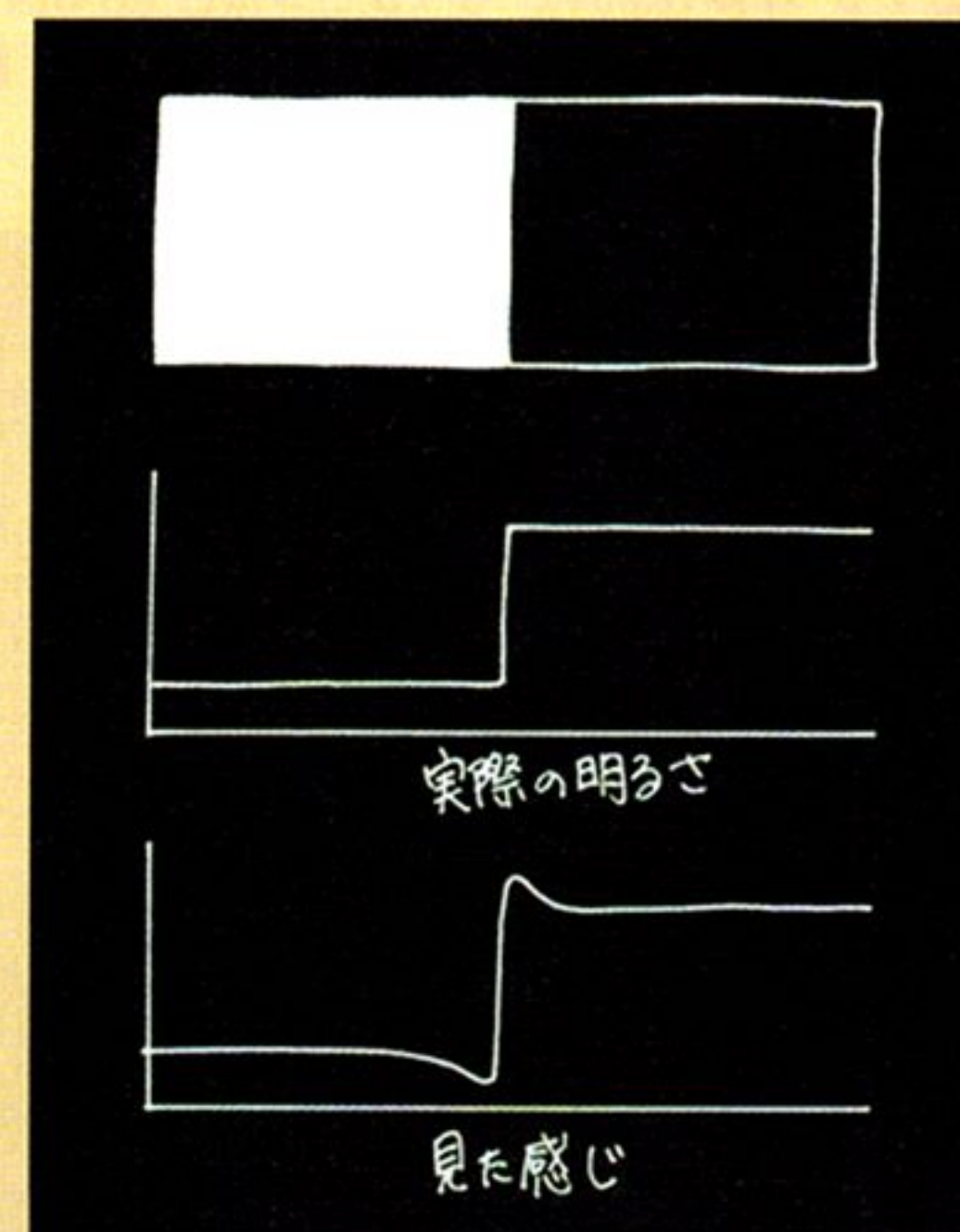
いまをさかのぼること約10年前（げ，もうそんなになるのか），X68000のC-TRACE68というソフトでレイレーシング画像を描いていた頃に「マッハバンド」という現象を知りました。X68000というマシンは，当時のパソコン（あ，ワークステーションか）としてはサービス過剰ともいえる65536色のグラフィック表示を誇っていました。これだけの色があれば，パソコンで扱う色としてはもはや十分なんじゃないかと思ったものですが，最近のパソコンが軒並み1600万色もの色を表示していることからもわかるように，実は6万色では人間の視覚を誤魔化すにはまだ少し足りませんでした。たとえば球面をレンダリングすると，データ上は滑らかな球面のはずなのに，X68000の画面に現れる球の陰影はグラデーションが滑らかにかからないのです。しかも，滑らかどころか洗濯板状にでこぼこがあるように見えてしまう，これが（場合によっては？）厄介なマッハバンドというものです。

幸いというか当然というか，C-TRACE68が生成する画像ファイル自体はフルカラーの24ビットデータだったので，いったんファイルに落としたデータを画面に表示するときにディザリングするプログラムを書いてマッハバンドを軽減していました。

人間の目は，明るさの変化する部分を強調して感じるようにできています。これを心理物理学では「辺縁対比」といいます。また，明るさだけではなく色の境界，たとえば赤と緑などでもこの視覚的不連続線は発生します（こちらは「同時色対比」）。ものの輪郭をはっきり認識するために我々が進化の過程で獲得した視覚の特性かもしれません。漫画など，輪郭線だけでものを描いても違和感がないのはこの現象に起因してるんじゃないかと僕は思っているのですが……（漫画の得意な日本人は，もしかして遺伝的に辺縁対比が強い？）。

面白いのは，この生理現象は，映像の情報が大脳に到達するより以前の網膜上ですでに起こっていることです。つまり網膜上の細胞たちは，隣りあう視細胞たちと情報をやり取りしているのです。図5に描いたように，錐体から脳へ伸びた神経系には何段階かのレイヤーが存在しており，視細胞が受け取った光は，生の情報のままで大脳に直接送られるのではありません。いくつかの加工が施されるのです。何段もの階層を隔ててから目的地の脳に達するなんて，なんとなくネットワークプロトコルに似てるなと思ったりしました。

実際の明るさ

見た感じ

辺縁対比

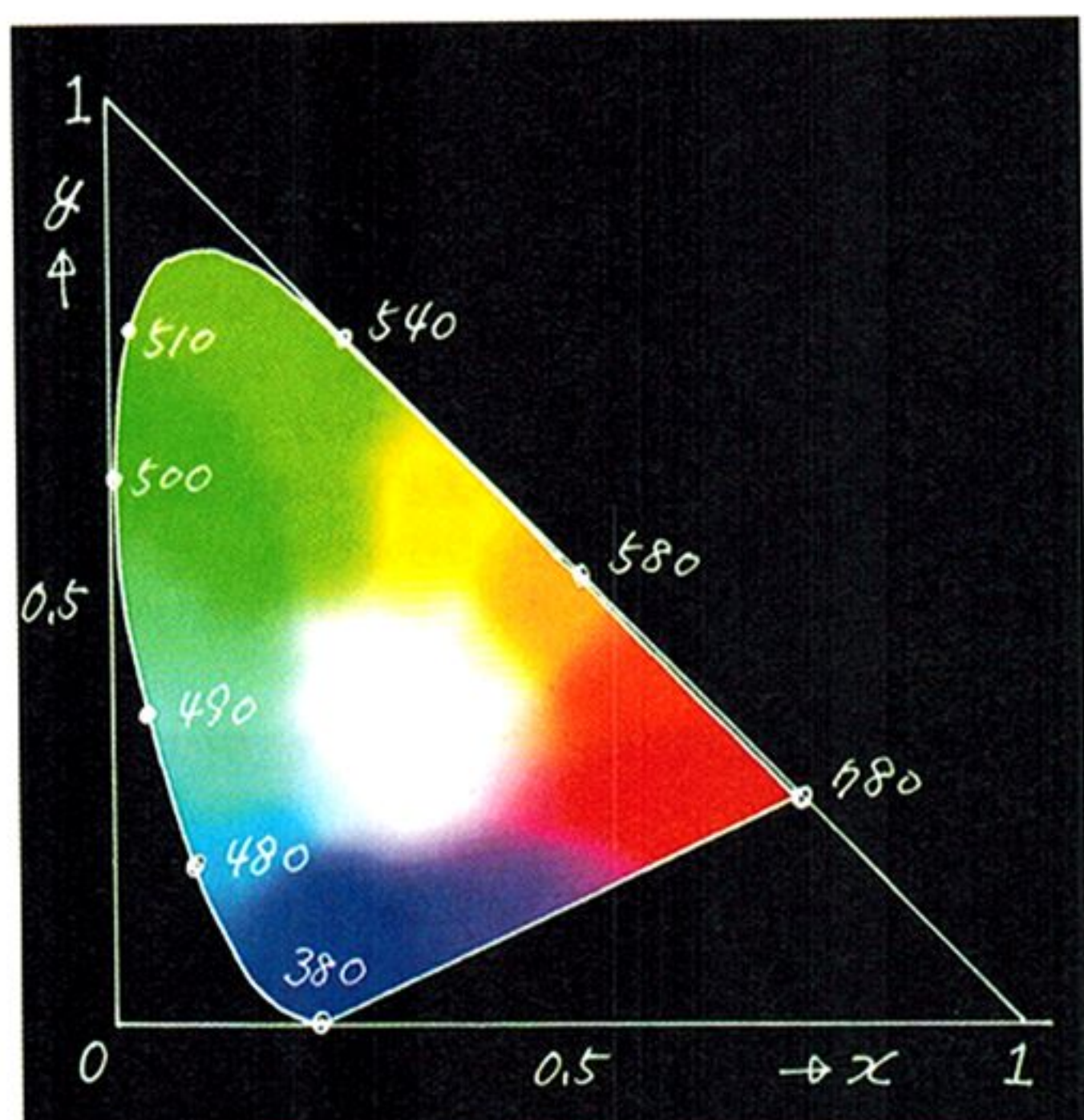



図5　CIEのxy色度表

RGB原色の知識がベースにあると，このヘンテコな形状の理由も理解しやすくなります。CIE色度表はお絵描き屋さんに必要な知識とも思えませんが，この形状ができた経緯を追っていくことで周辺の知識が得られて面白いかと思いますし，PC99の規格やCSSなどで採用されているsRGBなどもその基準はCIE色度表になりますので，教養ということで簡単に紹介しておきましょう。そういえばX11ではR5あたりからCIE xyによる色指定ができたっけかな。

純粋な刺激

RGBの3種類の錐体は，もっとも強く反応するのが赤，青，緑の波長なのであって，ほかの波長領域にも少なからず反応します。図6は，ヘルムホルツ（Helmholtz 1821-1894）という人が考えた感覚曲線のグラフを僕がアレンジしたものです（実測値ではないので山の高さなどはウソだが，概念的にはこんなものと考えて間違いない）。見てわかるとおり，Rの反応する波長領域ではGもBもわずかずつ反応しています。したがって我々が感じる赤のなかには，GやBの錐体の感覚も含まれているわけです。仮に，Rの錐体だけ反応させて，GとBの錐体の反応を遮断することができたら，どんなに鮮やかな赤が見られるでしょう。

同様に，GとBもほかの錐体の反応に汚染されているため完全に純粋な刺激を感じることはできません。特に緑などはRからもBからも大きく侵食されているので，明るいけれども薄い印象の色となっているようです。

現実の光では実現不可能ですが，RGBの錐体のどれかだけを興奮させる状態があると仮定して，それぞれの錐体が最高に反応した状態を頂点とした三角形の座標系で色を表現すると図7のようになります。三角形の重心の原理を用いることで，RGBの混色を表すことにしましょう。それぞれの反応の強さを重みとしたときに，それらの合成で感じられる色は三角形の重心に位置するわけです。すると，弓状の線はスペクトル上の単色に相当し，弦になる直線はスペクトル上にはないけれど，混色することで表現できる紫系の色を表すことになります。

弓と弦で囲まれた中は現実の眼に見える色で，それより外は，RGBのどれかの錐体だけ，もしくはその組み合わせだけが興奮した状態の，普通は見られない仮想的な色になります。

実際には見られないけれど，感覚的，原理的には存在するであろう色を仮定することで，「感覚的三原色」という概念を導入するわけです。この三角形の中で，人間が感じる可能性のあるすべての色を位置付けできるのがわかるでしょうか（ただし図に明度は表れてないよ）。

いまの話はちょっと置いといて，今度は「実際に眼に見えるRGB（スペクトル上の単色）」を「原色と決めた」ときに，それらをどんな割合で混ぜあわせればほかの色が作れるかということを研究したマクスウェル（Maxwell 1831-1879）という人の研究結果を見てみます。先のヘルムホルツの三角形と同様にRGB座標系を取ってグラフを描いています。その結論がだいたい図8のようになります。実際に見えるRGBが基準になってますから，三角形の頂点は，各RGBの錐体が汚染しあっている状態にあることがわかるでしょう。結果をプロットしていくと，スペクトル上の単色光がことごとく三角形の外を通ってしまったのです。

これはスペクトル上の純色は（目に見える）RGBの混色では表現できない

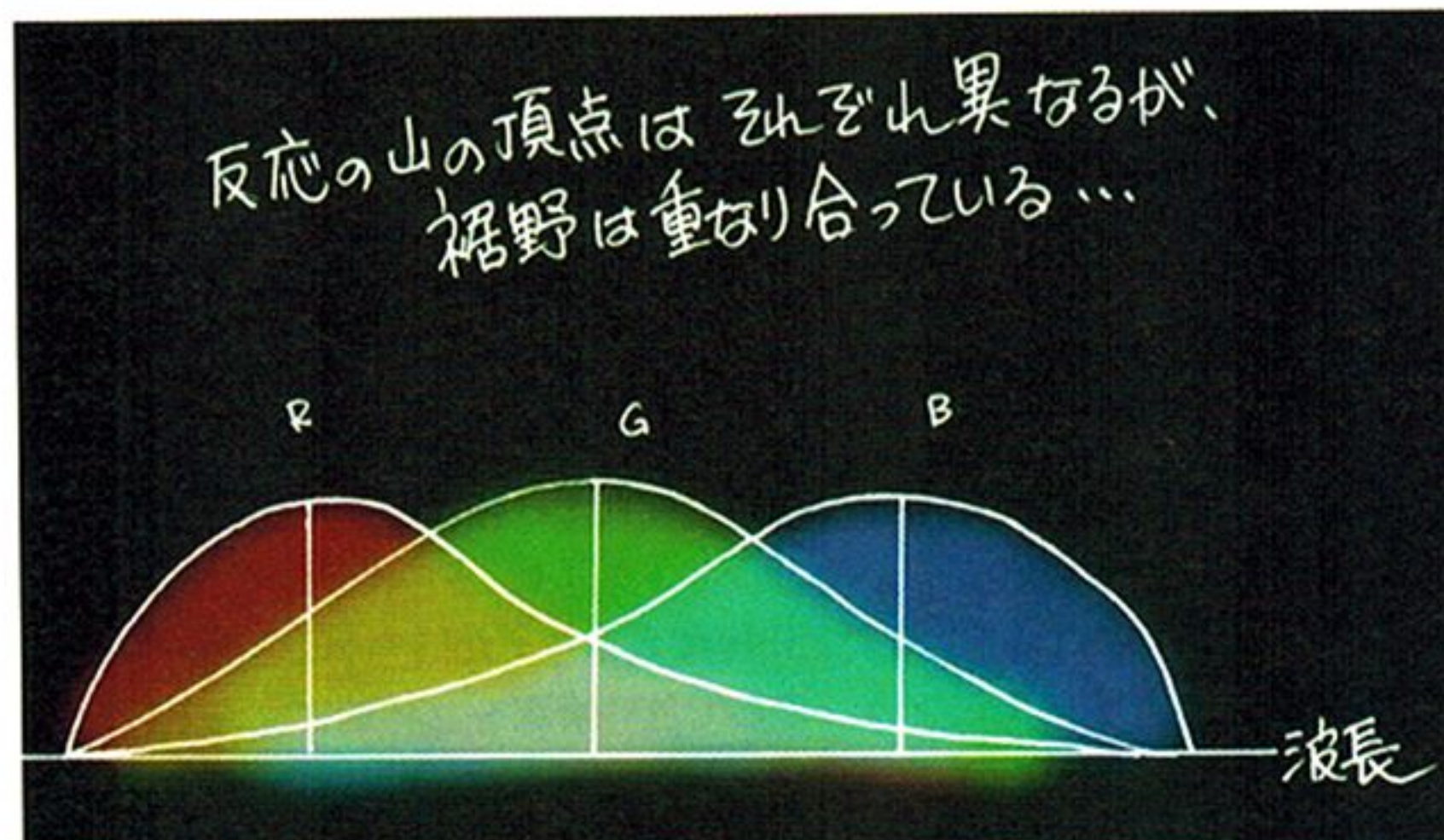



図6　ヘルムホルツのスペクトル基本感覚曲線をアレンジした図

図7

図8

ことを表しています。混色では「濁り」が発生するために，単色光の色の鮮やかさにはどうしてもかなわないのです。また，重心の原理から考えると，スペクトル上の点を表すにはRGBにマイナスのパラメータが必要なことがわかります。

色空間とベクトルの導入

さて，いままで表していたのは光の色度にだけ注目した座標系になります。実際には，光には明るさが関わってきますし，RGBという3パラメータで決定されるのですから，3次元の座標系を考えたほうがより詳しく色を表現することができるでしょう。そこで色の情報を立体的に立ち上げたものが「色空間」といわれるものです。

RGBを軸とする直行座標系で色空間を描いたものが図9のようになります。色を，RGBを成分とする原点を中心にしたベクトルととらえると，その色度は，色ベクトルが単位面と干渉した点で示せます。もちろん複数の色の合成もそれら色ベクトルの和で示せますし，その合成された色の色度も，合成ベクトルが単位面にぶつかった位置で一意に決まります。また，明るさはベクトルの大きさと考えると自然でしょう。

こんなふうに，ある色を示すためには色空間とベクトルの概念を応用すると便利になります。この色空間は，必ずしも直行座標系である必要はありませんし，用途が明確なら，アフィン変換だって自由にかけてしまって構いません。

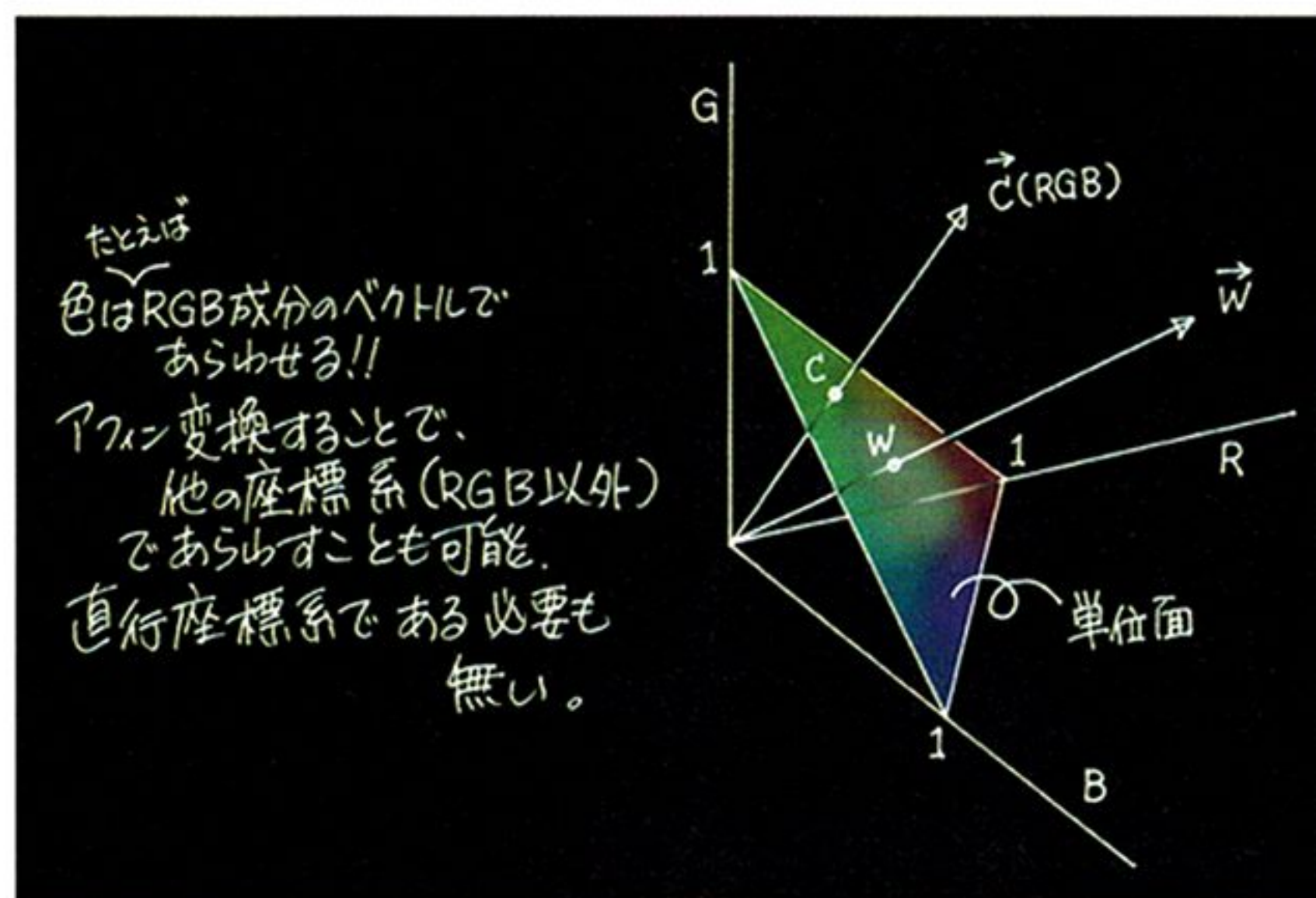
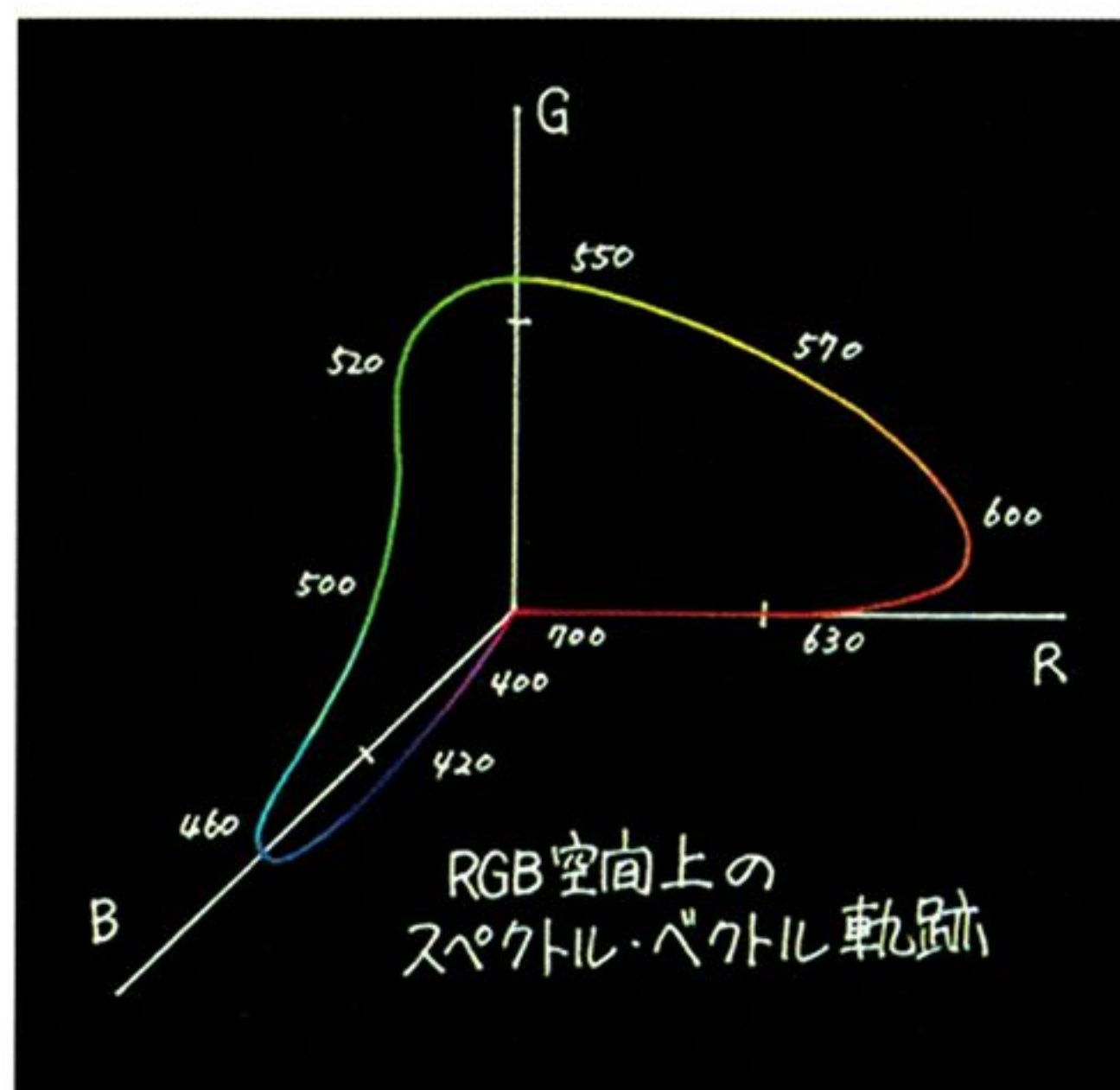
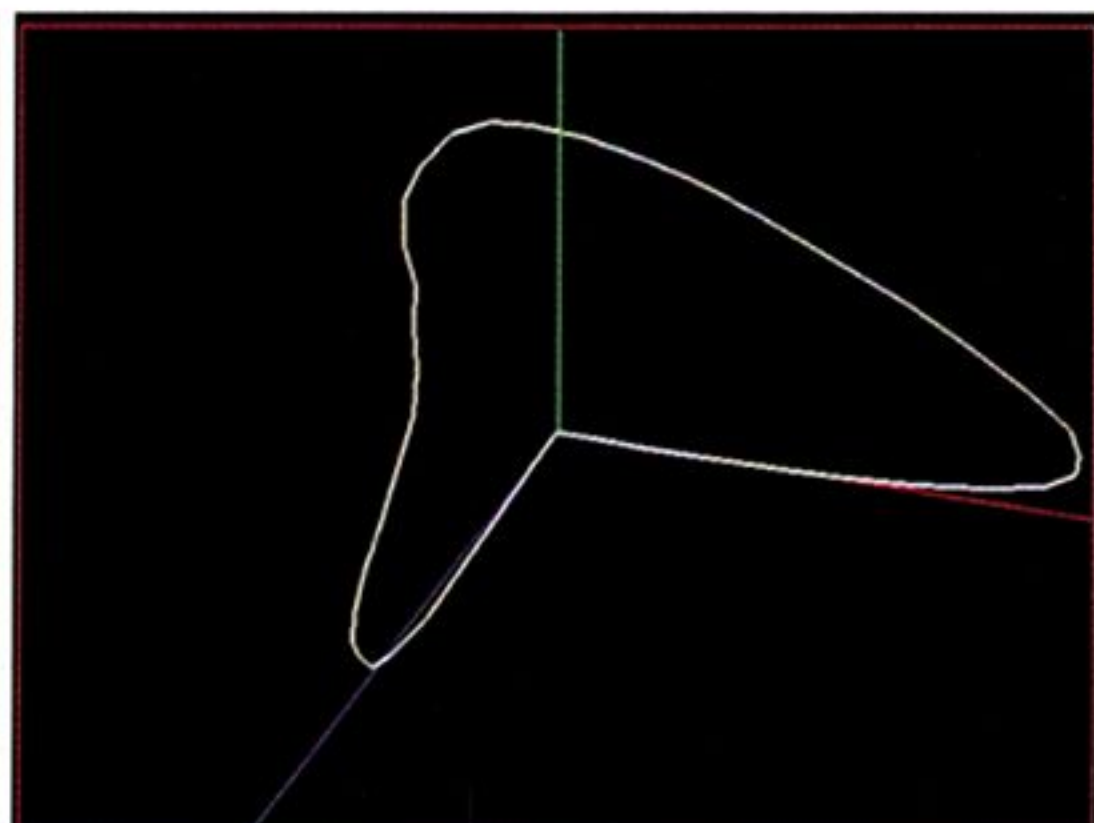
もう少しこのRGB直行座標系をいじくってみましょう。今度はスペクトル上の単色を，この色空間上にプロットしてみます。2次元の図で描くとイメージが湧きにくいのですが，だいたい図10のような形になります。始点400nmと終点730nmの両方を原点（0，0，0）に取った，ぐにゅ〜っと空間中を泳ぐ曲線になります。図6の感覚曲線を基底関数に取った3次元曲線といって間違いないかな？　ちょっとウソっぽいけど。このように，RGBを軸に取った座標系の中ではスペクトルの単色といえども単純な曲線に収まらず複雑な様相を見せます。うーん，なんか扱いづらいですね。

さて，色度すなわち色味を表すだけなら，単位面に注目するだけでよい

図9

図10

YC分離の話

Column

色の三原色がRGBだからといって「RGB原理主義」に陥る必要はありません。RGBはあくまで錐体レベルの物理的な構造であって，その組み合わせで起こる人間の感覚はもっと複雑で奥深いものです。「継時色対比」という色の残像現象から考え出された「ヘリングの反対色説（Hering 1878）」は，RGBよりももっと高レベルでの情報処理が視覚神経系で起きていることを示しています。継時色対比とは，赤い色を凝視したあとに白い紙を見ると緑の残像が，また，青い色を凝視したあとに白い紙を見ると黄色い残像が見える，つまり補色の残像が目に残ることをいうのですが，ヘリングはこの現象から，色の原色は赤緑黄青の4つだと考えました。もちろんRGB三原色の錐体が確認されるより前の話です（ベゾルト・ブリュッケ現象と呼ばれる現象からもこのことが裏打ちできるのですが，まぁとりあえずそれは置いておいて……）。

錐体の構造がわかる前に考案された論理だからといって馬鹿にするのは早すぎます。ヘリングの説は十分実用性のある原理で，視覚で実際に起きる現象なのです。マッハバンドのところで書いたように，網膜は脳に頼ることなく，ある程度独自に視覚情報をプリプロセスしています。ヘリングの反対色説も網膜上で起こる現象と説明され，R信号とG信号，そしてRとGで生成されたY（黄）信号とB信号の組み合わせが網膜内の水平細胞というところで絡みあって，R－G反応，Y－B反応が生じると考えられています。また，桿体の輝度反応であるW’反応や，R－G反応とY－B反応の合成であるW反応も別枠で存在します。

その生理的な現象を応用して考え出されたのが分YC離やYIQ分離という，映像方面で用いられる技術です。Yが輝度信号，Cが色信号を意味しています（YIQの場合はIが赤と緑，Qが青と黄）。人間の視覚は色の違いよりも明るさの違いに敏感なので，明るさ信号（Y＝主信号）に多くの情報量を割き，多少誤魔化してもできあがる映像の美しさにそれほど影響がない色の信号（C＝副信号）には少ない情報量を割り当てます。また，こうすることによって，白黒でしか表現できないデバイス（モノクロテレビなど）ではY信号だけをデコードすれば映像を表示できるので回路が単純になるのです。

色情報をIとQに分けているのは，多分もっと高次の（大脳に近い？）特性のためでしょう。人間の肌の色にヒトは敏感なことから，肌の色合いにもっとも寄与するI信号を次に重要な情報としたのだと思われます。一般的なテレビの色調整ツマミが緑赤なのはこのためです。

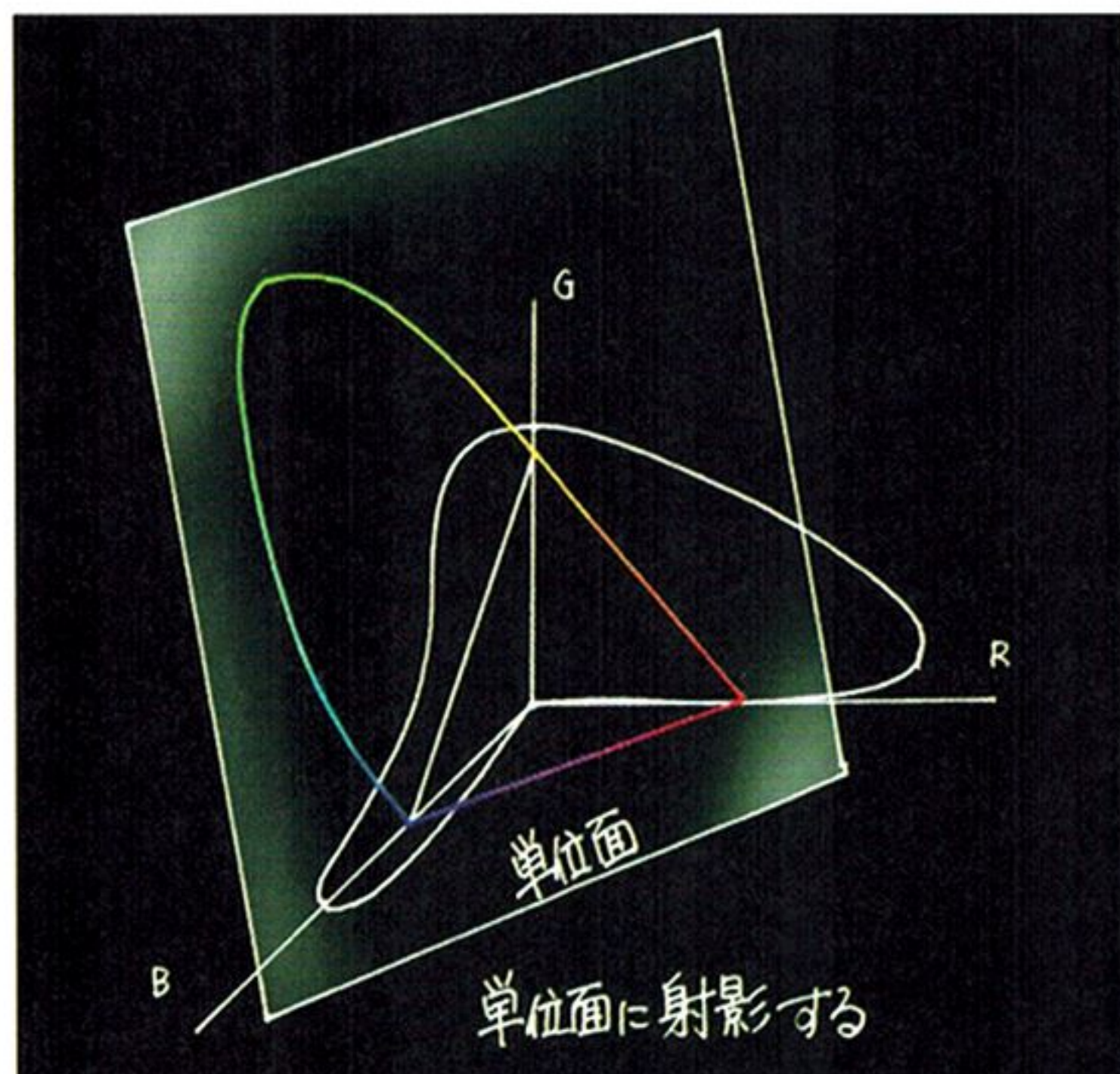



図11

ことは先の図9のとおりですので，この図10のスペクトル曲線も単位面に投影してしまいましょう（図11）。おぉ，なんとなく見慣れた形が現れてきましたね。

単位面に投影した結果が図12になります。これは図8のマクスウェルの図とほぼ同等の意味を表していますが，ずいぶんと不格好になってしまいました。

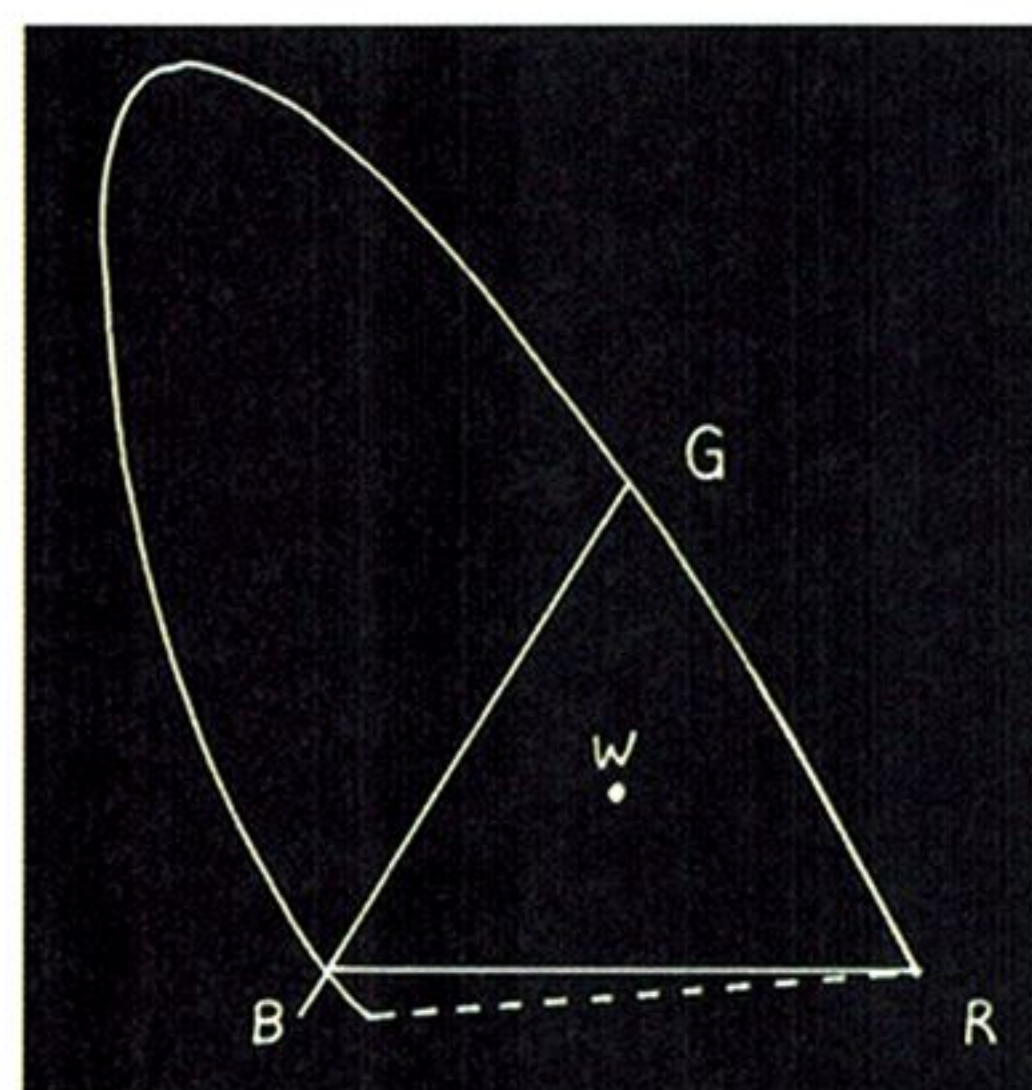



図12

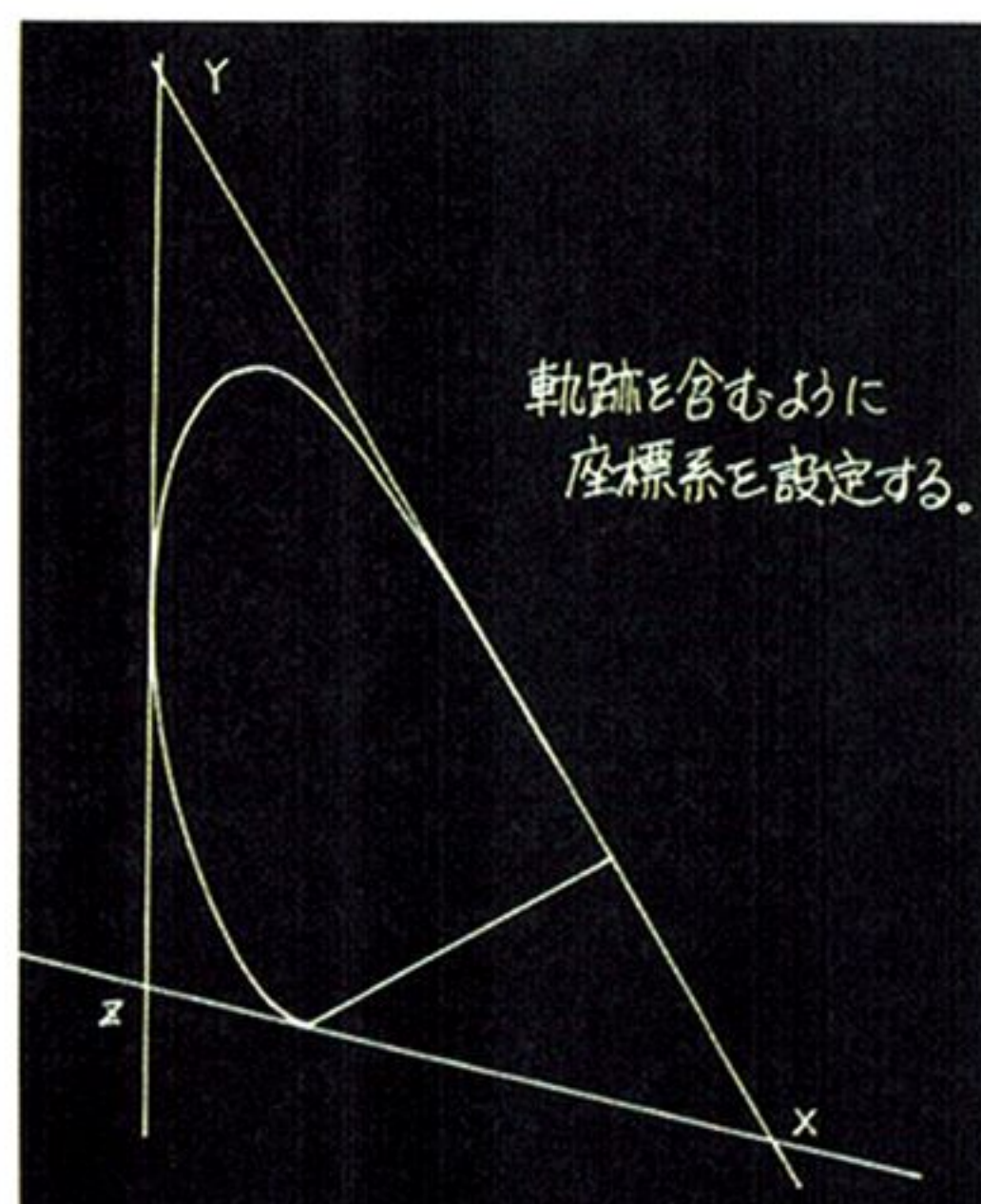



図13

スペクトル曲線が三角形の外側にはみ出ているということは，ベクトルに負の成分が含まれることを表しています。これはRGBの感覚が混じりっけがあって純粋ではないことが原因なわけですが，マイナスが出てくるのは計算上面倒くさいことがありますし，なんとなく気持ち悪いですよね。

それじゃ，負の部分をなくしちゃえ！　というわけで，原点を取り直したのが図13です。釣り鐘形全体を最小限に含むように原点を取ります。RGBではない別の空間に変換したのですから，これを仮にXYZ空間と名づけましょう。これがCIE xy表色系のもとになるのです

ここでは，もはや色の基準，すなわち原色はRGBではなく，XYZという謎の色になってしまっています。数学の虚数（imaginary number）と同じように，色にも「見えないけれども計算上現れる色」である虚色（imaginary color）という概念を導入したのです。先ほど出た感覚的三原色に似ていますけど，意味は違うことに注意してくださいね。

ちなみにZ-Xの線を紫赤の線と一致させないのが不思議ですが，これは「無輝線」という線にあわせた結果です。図には描いていませんが，色空間上に色はあるけれども輝度がない無輝面という（imaginaryな）平面が存在します。無輝面の概念は慣れないとちょっとわかりにくいと思うのですが，RGBがそれぞれ輝度を持っている場合に，その成分の合成でできた色の輝度は0になる可能性があるといえばなんとなく納得してもらえるでしょうか。無輝線は，無輝面と単位面の干渉した部分に発生する線です。

色度表の完成

さぁ，ここまでくればもうCIE色度表は目と鼻の先です。図13はXYZを結ぶ線がお互い斜めで扱いにくいので，こいつを直交座標に変換してしまいましょう。ZX軸とZY軸が直角に見えるように空間中で視点を移動したと考えてもいいかもしれません。はい，これでCIE色度表のできあがりです（図5）。

いままでの経緯を追ってくればわかるように，CIE xy色度表は，3次元の色空間上の一断面にすぎず，表しているのは基本的には色度です。また，X，Yは，色を指定するために作った，この世に存在しない仮の原色といった感じでしょうか（専門的には原刺激というらしい）。ただし次のような便利な特徴があります。

- 釣り鐘状の曲線（単色光軌跡）と底辺の直線（純紫軌跡）の内側に目に見えるすべての色が含まれる
- 色度をXYのパラメータで 表現できる。XYは正の値であり負にはならない
- XY面上の２つの色を混ぜたときには，それぞれの色の点を 結んだ直線上に混色が得られる
- 白の点を基準にして対称にある点同士の色が捕色になっている
- Y軸方向がその色の輝度を表している（無輝面がX軸に一致していることによる）
- 変換後，陰に隠れちゃったZ値は 実はZ＝１－（X＋Y）で表されている。そんでもってXは赤の度合，・Yは緑の度合，そしてZは黄色の度合を表している

などなど……。さぁこれからは「あの色」を示すのに，X，Yの数値を指定すれば済みますね。めでたしめでたし。

おわりに

というわけで，網膜と色空間のさわりを紹介してみました。プリミティブな話に終始してしまった感じもありますが，なぜ光の三原色がRGBなのか，や，三原色の混合で表現できる色は，広い色の空間の一部でしかないことがおわかりいただけたでしょうか。

絵やデザインに応用できる色の話としてなら，マンセル表色系，オストワルト表色系くらいは触れておいたほうがよかったかな。あとアブニー効果あたりも実用性があるかもしれないし，僕は勉強したことないけど，印刷やさん系の話も漫画に興味ある人には重要かもしれない，なんて，色の話は始めると，はっきりいってキリがないので今回はこのへんでおひらき。機会があれば，もっと絵寄りのことも書いてみたいと思います。

参考文献

金子隆芳〝色彩の科学〟1988/〝色彩の心理学〟1994 岩波新書
村上元彦〝どうしてものが見えるのか〟1995 岩波新書
Martin A.Fischler Oscar, Firschein〝Intelligence The Eye, the Brain, and the Computer〟Addison Wesley 1987
川上元郎〝新版 色の常識〟1987 日本規格協会
向井裕彦、緒方康二〝色彩学入門〟1996 建帛社

VBで作るフィルタリング実験環境

中野修一 Nakano Shuichi

２次元グラフィック処理でのフィルタ処理というのもすでにやり尽くされている感があるが，ここでは色空間を変えて，新しい次元での処理を模索してみたい。ありきたりのフィルタではなく，自分の本当にほしい効果を作るのはさほど難しいことではないのだ。

画像処理の実験をしてみよう。あいかわらずVB（VB5CCE）で行う（なんやかんやでVB6もインストールされてるんだけど）。

なお，今回は特集なんだけど，春号のテキストエディタについてちょっと。あの時点ではキー入力のたびにダミーキーを送り続けていたのだが，これはどう考えても頭がよくない。まず，普通の文字入力では障害は発生しないのだ。問題はカーソル操作のときだけなので，一般キーのときはスルーしてやればいい。さらにカーソル位置の不整合が発生するのはカーソルキーがこれまでの文字の流れとは違う向きになったときだけだから，それを判定してやれば負荷はかなり減る。その後，VBの特殊な仕様を利用した特殊なテキストコントロールを作ろうとしたのだが（文字の色などが任意に変更できる），特殊な仕様だけに使い勝手が悪く，思ったような処理が安定して実現できなかった。残念だ（いいバグだったのに）。

フィルタ実験環境

今回はVBで基本なフィルタ処理実験環境を作ってみた。なにをもってフィルタ処理というかなどを定義するつもりはない。ここではなんらかの加工を施すもの全般をいう。なにをやるかはまったく自由だ。「なんでもあり」でいける環境はちょこちょこいじれるVBなどが望ましい。もちろん，処理速度などを考えると本来VB向きではないんだが。

では，単純なところからいってみよう。2枚の画像の合成である。このプログラム全体のテスト用に作ったものだ。全体の構成とかはこの時点ですでに固まっている。ソースになる画像が2枚と出力結果ウィンドウ。たいていの処理はこれで足りるはずだ。

RGBはそれぞれ8ビット。24ビットカラーというやつで，32ビット整数型で扱われる。が，いちいちRGBを取り出すのは面倒なので，画像の読み込み時にそれぞれの配列を用意して処理中はもっぱらそれを使う。ただし，使用する変数は16ビット整数型だ。下手にバイト型を使うと符号付きの処理を行われたりするので使用していない。この辺は普通のinteger範囲の整数にlongを使うのと理屈は同じだ。処理時にはそれぞれの値にパラメータを加えていくことになるのだが，値が0以下になったときは0に，255を超える値は255に補正しないといけない。いわゆる飽和演算が必要となる。BASICには飽和演算型の変数はないので，演算後にいちいち修正しないといけない。これをif文でやっていくのは面倒なので，配列に飽和処理済みの要素を並べて参照している。要するに，－255～0は0に，1～255はその値，256～510は255にするようなテーブルを作っておく。もちろん，配列で参照するときに負の値はまずいので0～768の範囲にシフトしてある。速度的な効果は不明だが，気分的に楽になる。

RGBの取り出しでは，シフト演算は使えないので割り算を使うことになる。論理演算は有効なので上位ビットのマスクにはこれを使おう。剰余よりわずかに速い。実数除算は整数除算よりも高速だが，整数型に代入しても小数点以下をちゃんと切り捨ててくれなかったりするので整数除算を使用する。

合成など，足して2で割るだけの単純処理なので説明はしないが，だいたい以上のような下準備を行っておくとよいだろう。必要量の6倍のメモリを使うことになるが，せいぜい数MBならたいした量ではない。ピクチャーボックスからpointで逐次，値を取り出すようなことは非常に時間がかかるので避けたい。

ああ，なんかメモリの多い8ビット機でプログラミングしてる気分……。

HLS表色系を使う

さて，たいていの場合，グラフィックデータはRGBで扱われる。RGBはそれぞれ，0から255までの整数値となる。グラフィックツールの内部処理も当然，RGBベースとなっている。これには長所も短所もある。今回はHLS座標系での処理をメインにしてみたい。

HLSとは，色相，輝度，飽和度による色の表現形態だ。色の指定などでHSV形式を備えたグラフィックツールは多いが，今回はHLSを用いる。

要素のうち，LとSは0から255までの整数値となる。0から1までの実数値で扱っている文献も多いが，まあ8ビットの精度でもよいのではないかということで，ここでは色相を除いてすべて8ビット整数値を想定している。もちろん変数としては16ビットタイプを使う。

色相は0～360の範囲となる。これは円形に表示したときの度数そのものだ。0～359ではないのかという気もするが，参考文献で361段階だったのでそのままにしている。どっちでも大勢に影響はないのだが，ちょっと気持ち悪い。

さて，HLSを使うメリットはなんだろう。ひとつには，H，L，Sとパラメータの意味する内容が独立しているので，なんらかの処理を行う場合，RGB形式では同じ処理をR，G，Bについて3回繰り返さなければならないことが多い。ある種の処理では輝度だけが問題になるので，HLS形式なら，L要素だけ処理すれば済む。もちろん，表示のためにはRGBに戻さなければならないので，そちらのオーバーヘッドのほうが大きいようなら話は違ってくるのだが，たいていの処理では有効な手法となりうる（HLSではないが，輝度と色差を使ったYIQ形式の色空間でも同様なことがいえる。これらはRGBとアフィン変換可能であり，PC用ビデオカードではYIQからRGBへの変換をハードウェアで実装しているという点も頭の隅に入れておいたほうがいいかもしれない）。

ただ，そのために処理を行うごとにRGBからHLSに変換していたのでは割にあわないので，処理を始める前に（画像のロード時に）画像はRGB要素の配列とHLS要素別の配列に分解している。RGBとあわせて必要量の12倍のメモリを消費するが，数十MBくらいではたいした問題ではない。ちなみにこのプログラムでは3画面分で36倍分を使っている。BASICだと24ビットRGBデータからRGBそれぞれの要素を取り出すのも結構オーバーヘッドが大きい。こういう処理は最終的にCなりアセンブラで書くべきものだし，プロセッサパワーで解決するというのが昨今の流れだが，2Dのフィルタ系グラフィック処理というのは，もはやリアルタイムで処理されるべきものだと認識しておかなければならない。データ型もMMXで処理するときのことを考えておくのは当然だろうし，展開すべきものや省略すべきものなどを見極めておくことは大切だ。どっちみちBASICは遅いので高速化処理はできるだけ盛り込んでおいたほうが精神衛生上も好ましい。

とはいえ，HLSによる高速化というのは副作用のようなものにすぎず，HLS化の本来の目的は別のところにある。RGBで色を扱うのはどうも直感的ではない。人間の直感に近い色の表し方にHSVというものがあるが，これも向き不向きがある。HSVは色相，飽和度，バリューの3つの値で色を

図1　元画像

図2　アクセント処理で色のりをよくしてみた

示したものだ。色相は赤を0度以降，黄色，緑，シアン，青，マゼンタを60度ごとに円上に配置したときの色の系統の示す角度に相当する。飽和度は純色を255，灰色を0としたときの彩度に相当するものだ。バリューは，輝度と説明する人もいるが，正確ではない。たとえば，夜道にポスターが貼ってあったとしよう。そばにある街灯が明るければ，ポスター上の赤い文字も明るく見える。周りが最高に明るくなれば，文字本来の色がそのまま出てくる。バリューというのは，この周りの明るさに相当する値だ。

対して，駅ホームにある透過型ポスターなどを写真に撮る場合を考えよう。バックライトが明るければ本来の色に近くなるのは同じだが，最高にバックライトが明るくなると真っ白でなにも見えなくなってしまうだろう。値が0のときは黒で，255のときは白にまで変化する明るさがHLSのL値である。HSV表記でハイライト成分などを表現するときには飽和度を変化させていかなければならない。HLSなら光源の影響がパラメータひとつの処理で済むのだ。これらは透明絵の具と不透明絵の具で使い分けるのが正しいのかもしれない。また，光源を想定した写真的な処理に適しているのは明らかにHLSのほうである。

現存するほとんどのグラフィックツールではフィルタ処理をRGBベースで行う。そこで行われる「RGBそれぞれのバリューを上げる」というのと人間が想定する「RGBのそれぞれを明るくする」というのは必ずしも同じではない。

たとえば画像のコントラストを強くしたい場合，画像を構成する値の大きい部分はより大きく，小さい部分はより小さくという処理が行われる。これがRGBベースで行われるとコントラストが強くなればなるほど，色が濃い部分の彩度は上がることになる。「赤っぽい暗がり」は「べたっとした赤」に置き換えられる。普通の人が想定するコントラストの強い画像とはそういうものだろうか？　HLS系ではLの値についてのみ処理を行えばよい。彩度などはいじらずに済むので明暗だけのコントラスト処理ができる。

このように考えると，写真表現や光学的な現象のシミュレートなど，多くの分野でHLS表記は有効に使えることが想像できる。しかし，欠点もある。RGBとHLSの2つの色空間ではアフィン変換が効かない。そのため変換誤差も少なからずある。今回のサンプルで相互変換を繰り返さず，最初に分解してそのデータを使い回すのは変換誤差を少なくするという意味もある。

ちなみに変換部分はリスト1のようにしている。整数化し，色相不定時の処理を省いている簡略型だ。マウスカーソルの設定を2回やってるのは，もちろん1回ではうまくいかないことがあるからだ。差し障りがなさそうな部分は実数除算を使用しているが，速度差はほとんどないので，安心を取るなら整数除算にすべきかも。

基本環境はできたので，以下，サンプルのフィルタを順に見てみよう。

リスト1

```
Private Sub toHLS()
Dim m1, m2, cr, cg, cb, h, mm2, mm1 As Integer
Command1.Enabled = False
Command2.Enabled = False
For i = 0 To Picture1.ScaleHeight - 1
Screen.MousePointer = vbHourglass
    For j = 0 To Picture1.ScaleWidth - 1
    c = Picture1.Point(j, i)
    bb0(j, i) = (c And 16777215) ¥ 65536
    gg0(j, i) = (c And 65535) ¥ 256
    rr0(j, i) = c And 255

    r = rr0(j, i): g = gg0(j, i): b = bb0(j, i)

    m1 = max(r, g, b)
    m2 = min(r, g, b)
    mm1 = m1 + m2
    mm2 = m1 - m2
    l0(j, i) = mm1 ¥ 2
    If m1 = m2 Then
      s0(j, i) = 0: h = 0
    Else
      If l0(j, i) < 128 Then
        s0(j, i) = mm2 * 255 ¥ mm1
      Else
        s0(j, i) = mm2 * 255 ¥ (510 - mm1)
      End If
      If r = m1 Then h = (g - b) * 255 / mm2
      If g = m1 Then h = 510 + (b - r) * 255 / mm2
      If b = m1 Then h = 1020 + (r - g) * 255 / mm2
      h0(j, i) = h * 60 / 255
      If h0(j, i) < 0 Then h0(j, i) = h0(j, i) + 360
      If h0(j, i) > 360 Then h0(j, i) = h0(j, i) Mod 361

    End If
  Next
Next

Screen.MousePointer = vbDefault
Command1.Enabled = True
Command2.Enabled = True
Screen.MousePointer = vbDefault
End Sub

Private Sub HLStoRGB()
Dim m1, m2, cr, cg, cb, mm0, mm1 As Integer
    If l < 128 Then
      m2 = l * (255 + s) / 255
    Else
      m2 = l + s - l * s / 255
    End If
    m1 = 2 * l - m2
    If s = 0 Then
      r = l: g = l: b = l
 '     gg1(j, i) = g: bb1(j, i) = b
    Else
      r = hrgb((h + 120) Mod 361, m1, m2)
      g = hrgb(h, m1, m2)
      b = hrgb((h + 240) Mod 361, m1, m2)
  '    gg1(j, i) = g: bb1(j, i) = b
    End If
End Sub

Private Function hrgb(hue, n1, n2)
If hue < 60 Then hrgb = n1 + (n2 - n1) * hue / 60
If hue > 59 And hue < 180 Then hrgb = n2
If hue > 179 And hue < 240 Then hrgb = n1 + (n2 - n1) * (240 - hue) / 60
If hue > 239 Then hrgb = n1
End Function
```

図3 元画像の輝度分布た

図4 輪郭処理後の輝度分布

HLS test

では，HLS系でのフィルタ処理を書いてみよう。まずは，それぞれの値になにか足したり引いたりするだけの処理だ。スライダで値を強くしたり弱くしたりできる。原理はなんということのない処理だが，もともと意味のあるパラメータなので，効果は侮れない。Hについては色相の変移，Lについては，正しい輝度調整（真っ黒から真っ白まで調整できる），Sについては彩度調節となる。

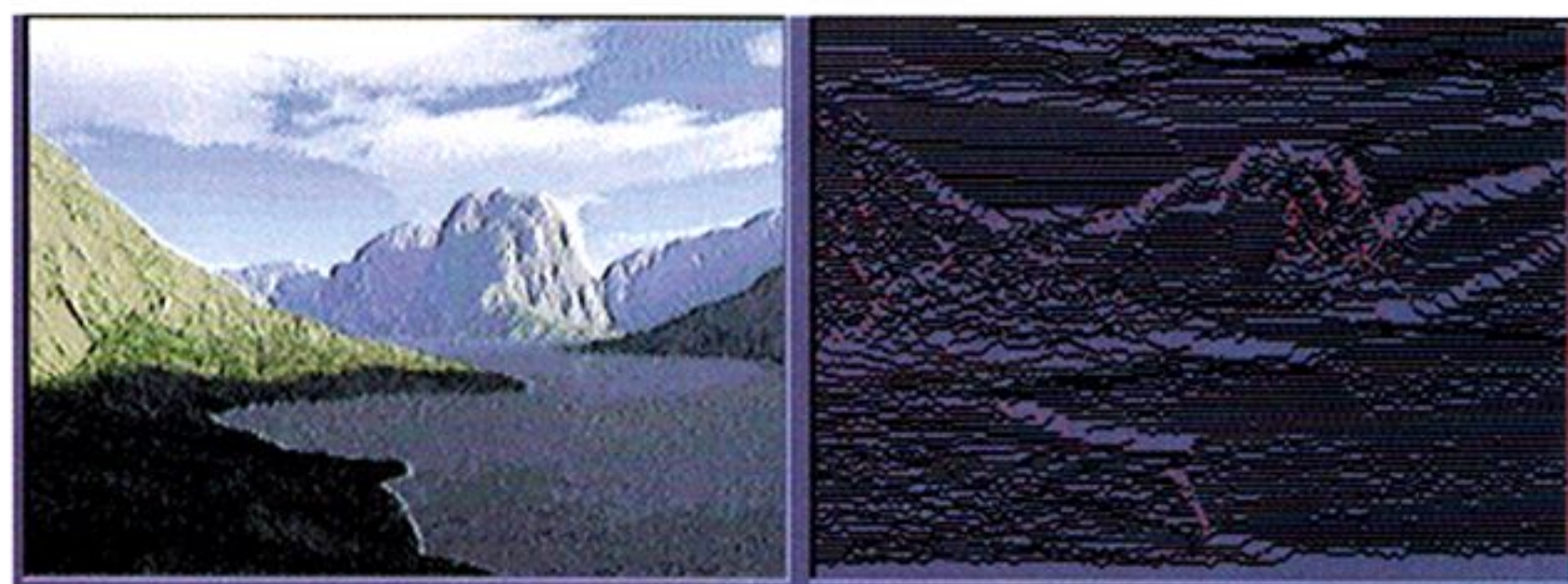
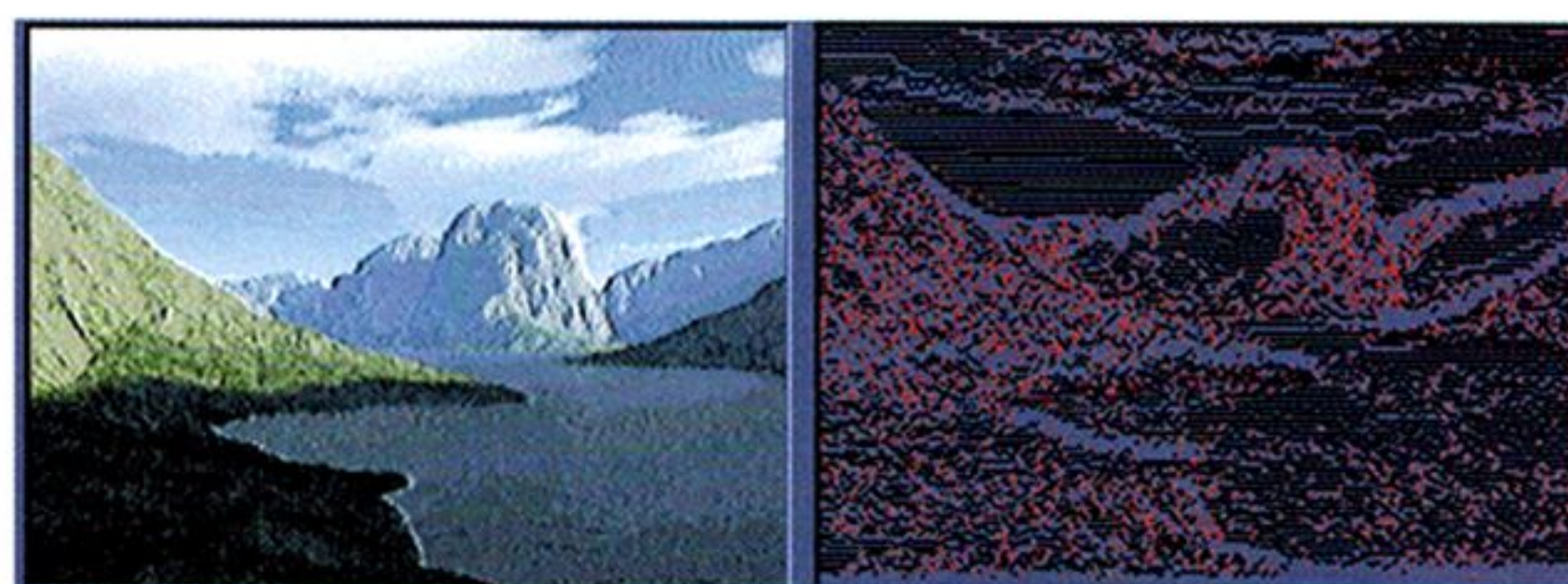



図5 擬似的なモーションブラー効果

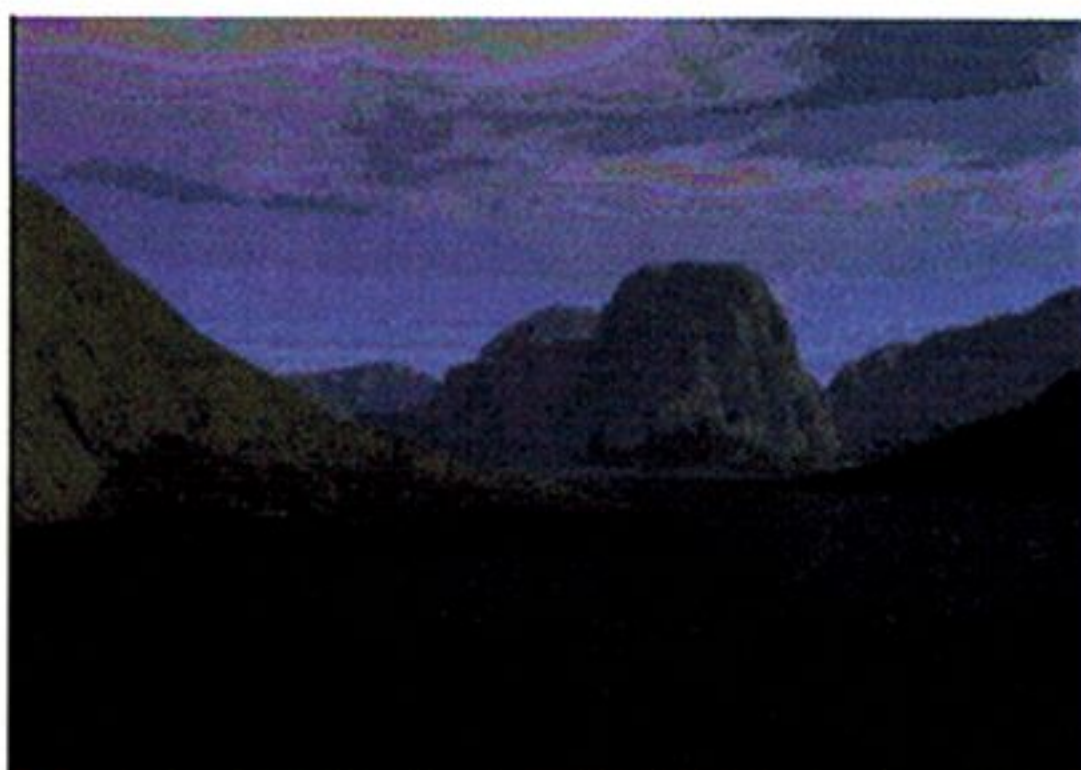



図6 環境マッピング型バンプマッピング

HLS accent

次に先ほど例に挙げたコントラスト処理を試してみたい。コードについていちいち解説するより結果を中心に見ていこう。Lの値をいじることで期待どおりの結果が得られた。Photoshopなどのコントラスト調整を思い切りかけた画像と見比べてほしい。

これだけだとつまらないので，L以外の要素について処理を行う。Sだと彩度が高い部分はさらに高く，低い部分はさらに低くなどという処理になる。実際にやってみると彩度を下げる処理のおかげでたいていの絵は破綻してしまう。これだと特殊効果用だ。面白くない。そこで，逆の処理を行ってみる。コントラストを下げるとどうなるだろうか。全体の彩度は中域に集中する。これは多くの場合，画面全体の彩度を高めることにもなる（もちろん絵によるのだが）。色のつながりはよくなるので強くかけても破綻はしない。色味が足りないんだけど，彩度をかさ上げすると下品になるのでヤだという場合には有効。

さらに色相にいってみよう。0に近いモノはより0に近く，360に近いモノはより360に近く……なんてやってても意味がないので（0も360も同じ赤なのだ），処理内容から考え直そう。感覚的にとらえ直すと，純色に近くなるのが正しいだろう。彩度とは別の意味でRGBの純色に近づける処理を考える。RGBはそれぞれ0，120，240に相当するので，間にある色はそれぞれに引きつけてやればいい。処理結果は色自体はかなり変わる可能性があるもののだいたいクリアな色味になっていくはずだ。逆も考えよう。中間色の色相に近づける処理だ。結果の評価は人それぞれだろうが，穏やかな色調にまとめるという処理でよいのではないかと思う。

EnSharp

本来2次元で行うべきものだが，もっとも単純化した例を実装している。つまり，

次のドットと輝度を比べる
2つのドットのうち明るいほうをより明るく暗いほうをより暗くする

という処理を行う。ただし，変化量の少ない部分は処理をしない。

ビデオのエンハンサと同じでエッジ部分に白い線が出たり，画像としては上品ではなくなるのだが，エンハンス過多な絵作りを「解像度が高い」といって珍重してくれる人もいるので世の中はわからない。

一般にデジカメ画像などには，ある程度輪郭補正（アンシャープマスク）を行ったあとでやや縮小するというのが効果的だ。

Analysis

では実際の絵の輝度変化はどんな感じになっているのかということで，画像のL値を高さに置き換えて簡易的に表示したものが図3，4である。

前ドットとの明るさの差が一定のスレッショルド以上のものを見てみよう。3番目のスライダが0以外の場合には，差分値が指定値を超えた場合に赤いドットで表示されるようになっている。

Blur

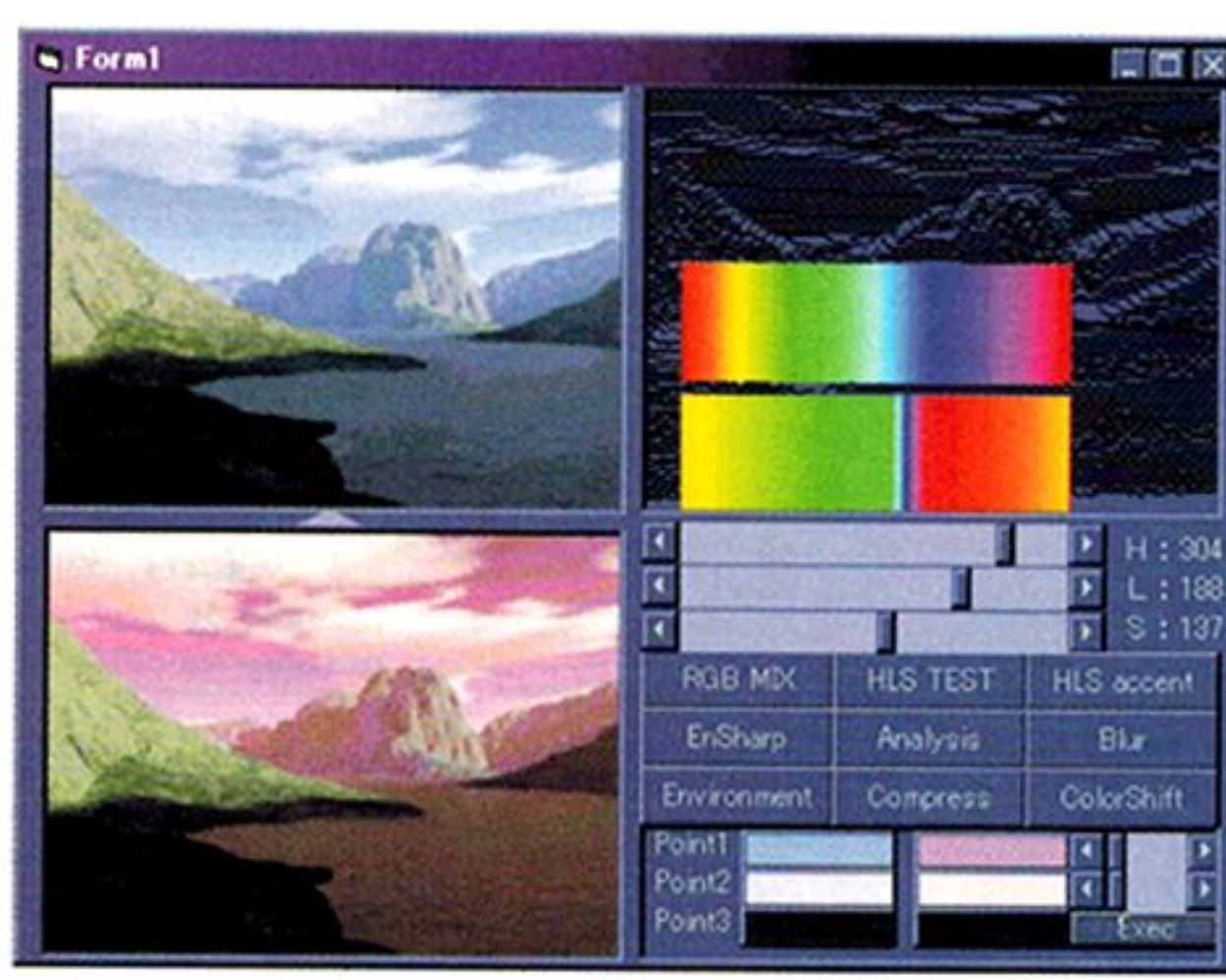
昔，低コストでモーションブラー効果を得るにはどうしたらいいかとか，横内君と話していたときに示したもの。後処理で流線を加えてはどうかというものだ。全画面モーションブラー効果だと負荷制御ができないので，ドット制御としてみたわけだ。消失点は随時変えるとしても効果はいまいちか。エッジだけ抽出して消失点からぼかすってやつのほうがマシかも。まあ，まともにやってもできてしまうのならそれがいちばんだが。

図7 色の圧縮処理

図8 任意の色の色相を変換する

Environment

L値配列をこの画像のハイトマップと見なし，Environment型バンプマップを試みたのが図6だ。

そのドットの傾きに相当する法線ベクトルの延長と背景球の交点から背景球の要素を取り出す。もちろん三角関数を使って計算したりなどはしない。こういう一意に決まる値は，あらかじめ計算してテーブルに持っておくだけでよい。X，Yそれぞれの取りうる傾斜は，整数値で－255～255の範囲にすぎない（そうなるように設計している）。環境球との

交点座標は……などということもせず，単純にx，yのディスプレースメントに変換している。光源計算しないというアルゴリズムがなかなか素晴らしい。描画するドット数が512ドット以上であればテーブル参照のほうが高速だ(今回は40592ドット描画している)。非常にいいかげんなのだが，なんとなくそれっぽいのでよしとしよう。

Compress

この処理はちょっと説明しにくい。一定の帯域でカットオフして圧縮してレゾナンスをかけて出力している。はっきりいうとオーディオフィルタの処理である。

ディストーションを自然に入れるにはどうするかとか，エコーをかけてみるとどうなるかとか，FM変調できないかとか，いろいろやってみて無難なものがこの辺りという感じだ。これはいわばフィルタリングのためのフィルタだ。きっとなんらかの効果は出るが，どうやって使うかなどは考えていない。フィルタは入力された信号に対して一定の操作を行ってデータを返すもの。とりあえずいろいろ作ってみて使い方はあとから考えればいい。

ColorShift

色変換で使用頻度が高そうなものを作ってみよう。色補正用のフィルタだ。

任意の3つの色を指定した色に変換しつつ，色空間全体を変換していく。写真を撮って，肌色を補正しつつ花の色は変えないとか，黒を沈めたままで全体に少し明るくし白はさらに伸ばすとか，そういったリニアでない補正ができる。

とりあえず，画面から3点の色を拾ってきて，それぞれの色相を任意に変換する。これで一部の色相範囲だけを変化させることが可能だ。補正はHLS空間で行っているが，Hを基本とし，L，Sは従属的な扱いで制御している。色相以外に輝度や飽和度をちゃんと補間してもいいのだが，あまり意味がないような気がして実装していない。なお，この処理はHLS系でやるのも興味深いが，RGBモードはつけておくべきだろう。色相が圧縮されると妙に色の変化が不自然になる部分もある。

また，色指定で本来あるべき順番がクロスしたりすると補間がうまくいかないので不連続な色相パターンになる。が，それはそれで面白そうなので，データチェックは入れていない。妙に離散量になるバグもあるのだが，これも面白いのであまり深く追求していない。

HLSの可能性 ── ぼけ味を見る

せっかくなので，フィルタとして書いてはいないのだが，HLSを使った例を見てみよう。EX for Winのぼかしフィルタだ。

「ぼけ」というと，やはり写真のぼけ味を出したいもの。が，グラフィックツールのぼかし処理で行うとなんとなく違ったものになる。

グラフィックツールでのぼかし処理については説明は必要ないだろう。近傍の画素と平均化した色に置き換えていく処理だ。距離に応じて比重をつけたものを(常識的にはつけるのが当たり前だが)ガウスぼかしと呼んで区別する処理系もあるようだ。

カメラでは背景のぼけはどういう風に起きているのだろうか？

焦点に結ぶべき画像が広範囲に広がってしまうというのはぼかし処理と同じ理屈でよいのだが，フィルムなりCCDの特性を考えないことには同じ処理は再現できない。CCDを例に取ろう。CCDでは受光した光の量に応じた値の電圧を返す。なにもないときは真っ暗で光の量に応じて値が変化する。あとはガンマ特性だかなぜだか不明だが，実際の写真を見るとピンボケの際には明るい部分の広がり方のほうが暗い部分よりも大きくなっている。原理上，暗い部分がピンボケによって「より暗く」飽和することはありえないが，明るい部分がより明るく飽和することはいくらでもあるということだろうか。

そこで，ぼかし方にひとつのルールを与えよう。

「明るい部分ほど広く周囲の画素に影響を与える」

これだと暗い部分はあまりぼけないが，明るい部分はよく広がる。このルールで処理したものが図である。実際にカメラで撮影したものと比較するとまだ違いがあるように思われるが，傾向としては近づいた感がある。

整数型配列に入れた値を取り出すと浮動小数点が残っていたり，255.999999あたりの数値を普通に整数化すると256になってしまっていたり，VBはあいかわらずワンダーに満ちている。ひょっとして私が想定していたメモリ量の4倍くらいを使ってるかもしれないなあ……。いちいち型変換するのがいいんだけど，確実に型変換関数を呼び出して処理するのだろうしなあ……。

図9　実際のカメラでピンをはずしてぼけの具合を確認

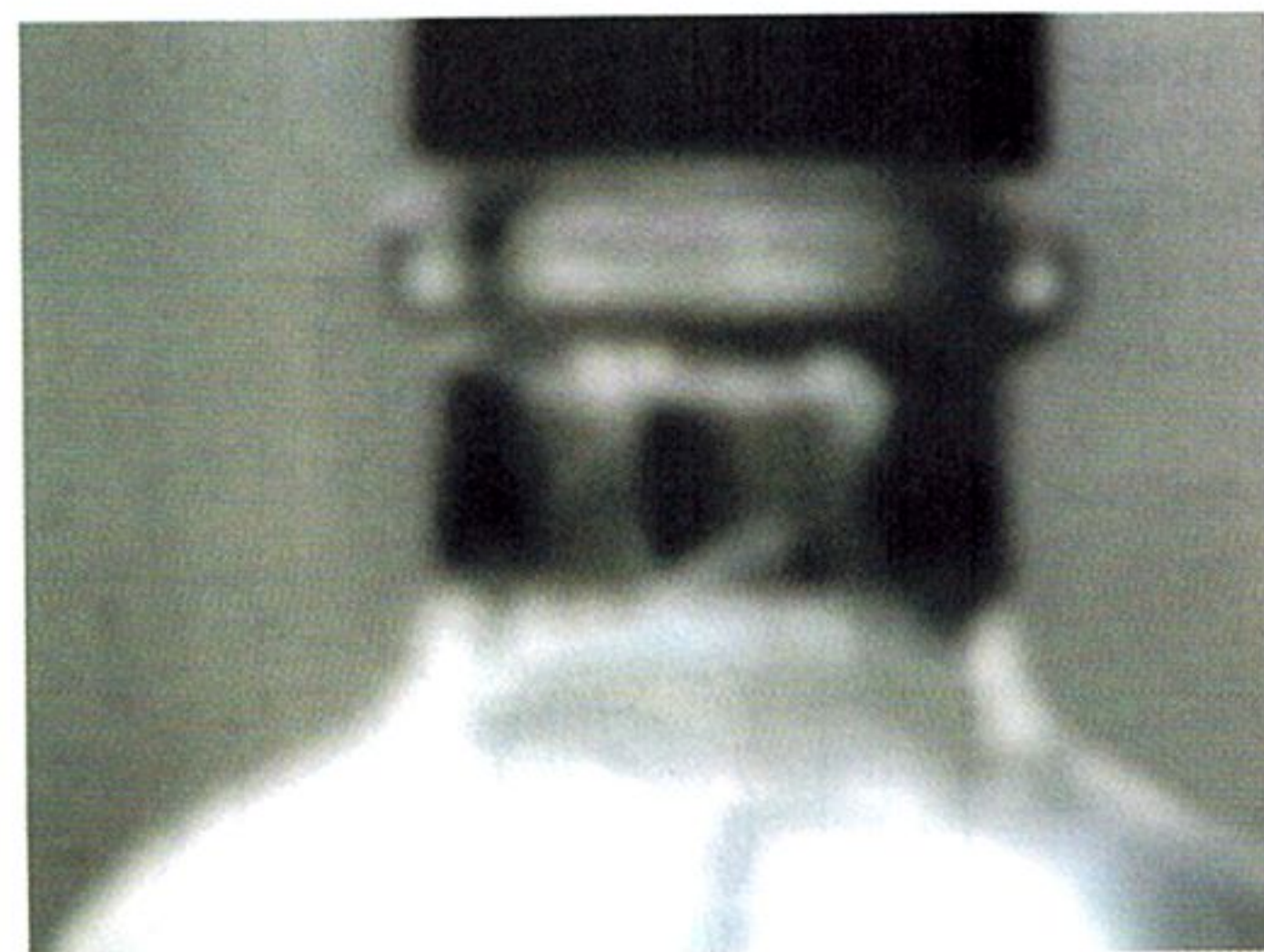



図10　グラフィックツールで思いっ切りぼかした画像

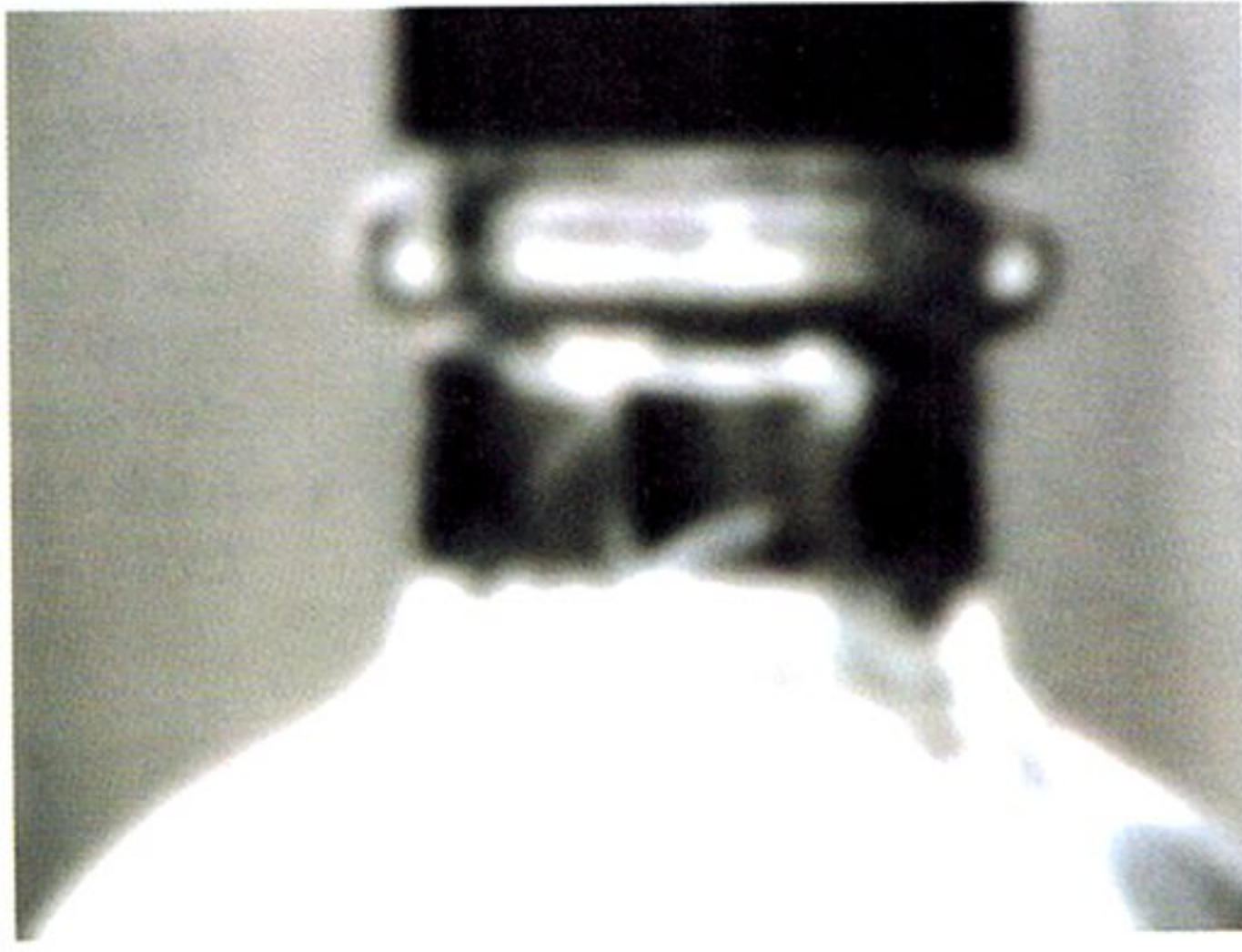



図11　輝度補正ぼかしを加えた画像

EX for Win用プラグインの作成

菊地 功 Kikuchi Isawo

旧Oh!X読者ならばグラフィック拡張システムからペイントツールへと進化したEX-Systemをご存じだろう。そのWindows版もぼちぼちと展開されている。ここでは現行バージョンのプラグイン作成について解説する。

本誌をご覧の読者には，EX-Systemをご存じの方も多いと思う。その昔，Oh!X誌上で作成されたX680x0用のグラフィックツールである。しかし，そのEX-SystemのMookに付属のCD-ROMに，EX for Windowsがこっそりと収録されていたことは，あまり知られていない（かもしれない）。その名のとおり，EX-SystemをWindows用に移植したもの，ではなくて，まったく別に作り始めたグラフィックツールなんである。これがひっそりと筆者のホームページでバージョンアップを繰り返し，やっと完成，なんてまだまだほど遠い状態である。というか，ここ1年くらいろくにいじっていない。まったくもって困ったちゃんなのであるが，その理由は次のようなことが挙げられる。

1）バージョンアップしても，ひっそりと自分のホームページに公開するだけなので，張り合いがない

2）ギャラが出ない。もともと筆者の趣味で始めたことだが，やっぱ霞だけを食って生きてるわけにもいかないので優先度は低くなる

3）飽きた

本来ならば，もうとっくに形になっていなければならないのだが，上記の理由により，EX-SystemのMook版からそれほど進化はしていない。完成すれば，またMookにできそうなのだが，Mookってのは特に締め切りがない（完成したときが締め切り）ので，なかなかその気にならない。なにせ，切羽詰まらないと始めない怠け者なもんで（この原稿も）。ちょっと触らないとなにしていたのか忘れ，さらに触らないと以前の状況を思い出すのに時間を費やし，さらに触らないと存在そのものを忘れてしまう。確か，現在は裏画面をつけようとして，ハマって身動きが取れなくなっていたような……。

ただ，Oh!Xも復刊したことだし，ここでまた紹介できるようになると，少しは悔い改めてやる気になるかもしれない。というわけで，今回は本体は置いといて，プラグインの作成法について説明したい。といきなりいっても知らない人はわからないだろうから，ちょっとだけ本体の説明をしておこう。

EX for Windows

わりと旧式で基本的なペンツールしか持たないグラフィックツールである（図1）。だから，フォトレタッチとか呼ばずに，グラフィックツールなんである。ただし，ペンの速さにはちょっとだけ自信がある（っても，C++で書いてんだけど）。開発を始めたのは486DX2/66マシンであったため，ぶっといブラシとか使っても，そのクラスで十分実用になるように作ったつもりである（すでにそのマシンはバラバラなため，検証はできないが）。特徴は，8ビットのマスクを備えていること。αといったほうがわかりやすいかもしれない。要するに，マスクされている部分の上から描画を行うと，そのマスクのレベルに応じて半透明色で描き込まれる。マスクはカラーと同様にパレットから強度を選択し，ペンで描き込むことができるのだが，それがむしろ使いにくいという意見もある。

あと，コピーペーストのときなど，カラーとマスクを任意選択してさまざまな合成を行えるのだが，はっきりいって慣れないうちは制御できない。作者自身も，「あそこは確かこういうコードを書いたはずだから……」とか考えてしまうことがある。まあ，慣れればどうにかなるとは思う。

また，本体側の機能を拡張するために，プラグイン方式も採用している。このプラグインはPhotoshopのプラグインと完全互換，なはずがなく，独自の方式である。というよりは，本体側のDIBのメモリアドレスやカレントカラーを取得するといった，低レベルでかつ「必要だからつけた」的な安易なAPIが揃っている。Photoshopのプラグインがどのようになっているかは知らないが，要するにプラグインっぽくなく，悪さでもなんでもできてしまうオープンな方式なんである。今回はこのプラグインを自分で作って，EXの機能を拡張しようというお話だ。EX自体の使い方はヘルプがついているので，そちらをよく読んでもらいたい。

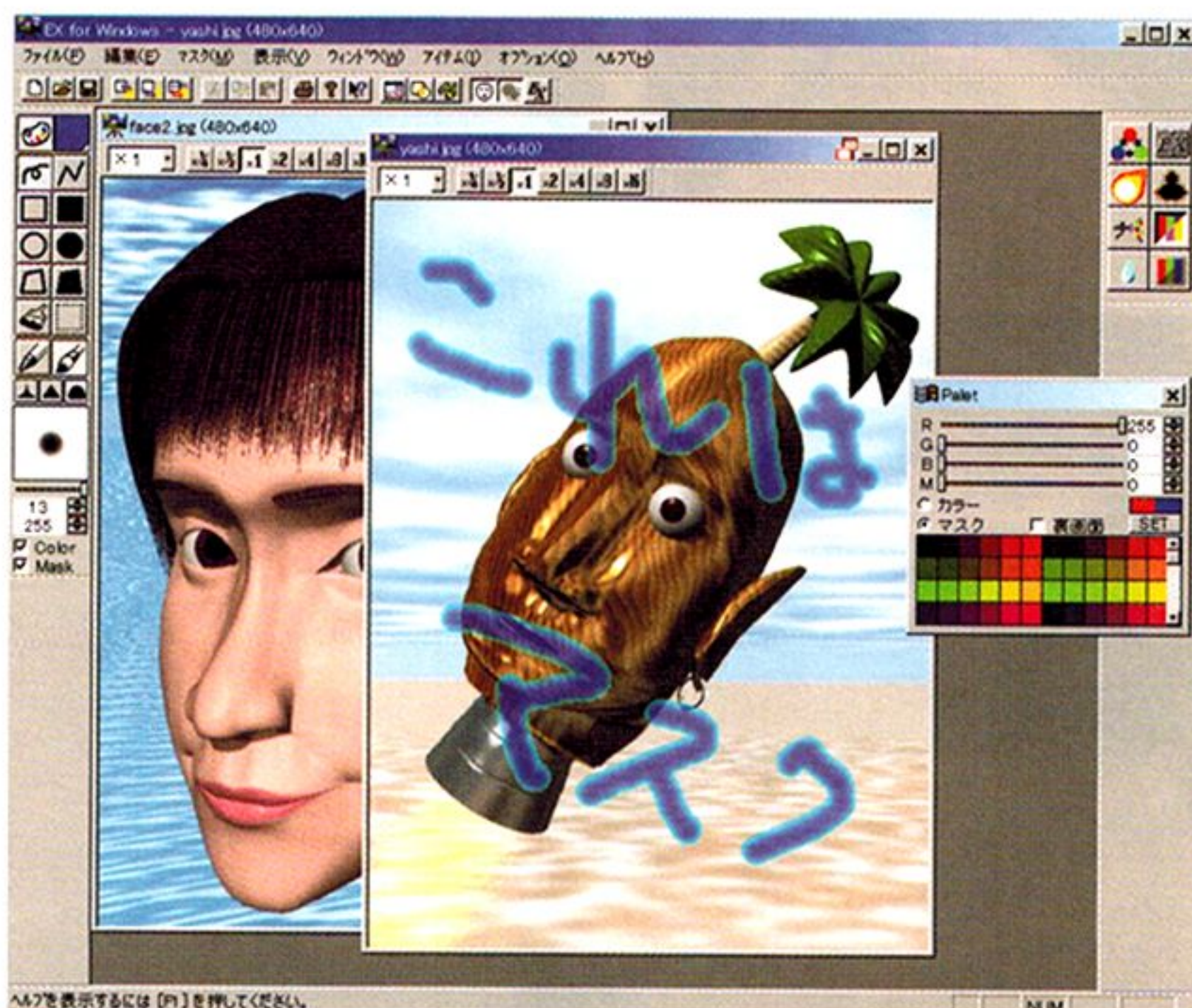



図1　EX for Windowsの外観。青く表示されている文字がマスク。ちなみにこれはハマっている途中開発バージョンなので，収録されているものとは若干違う

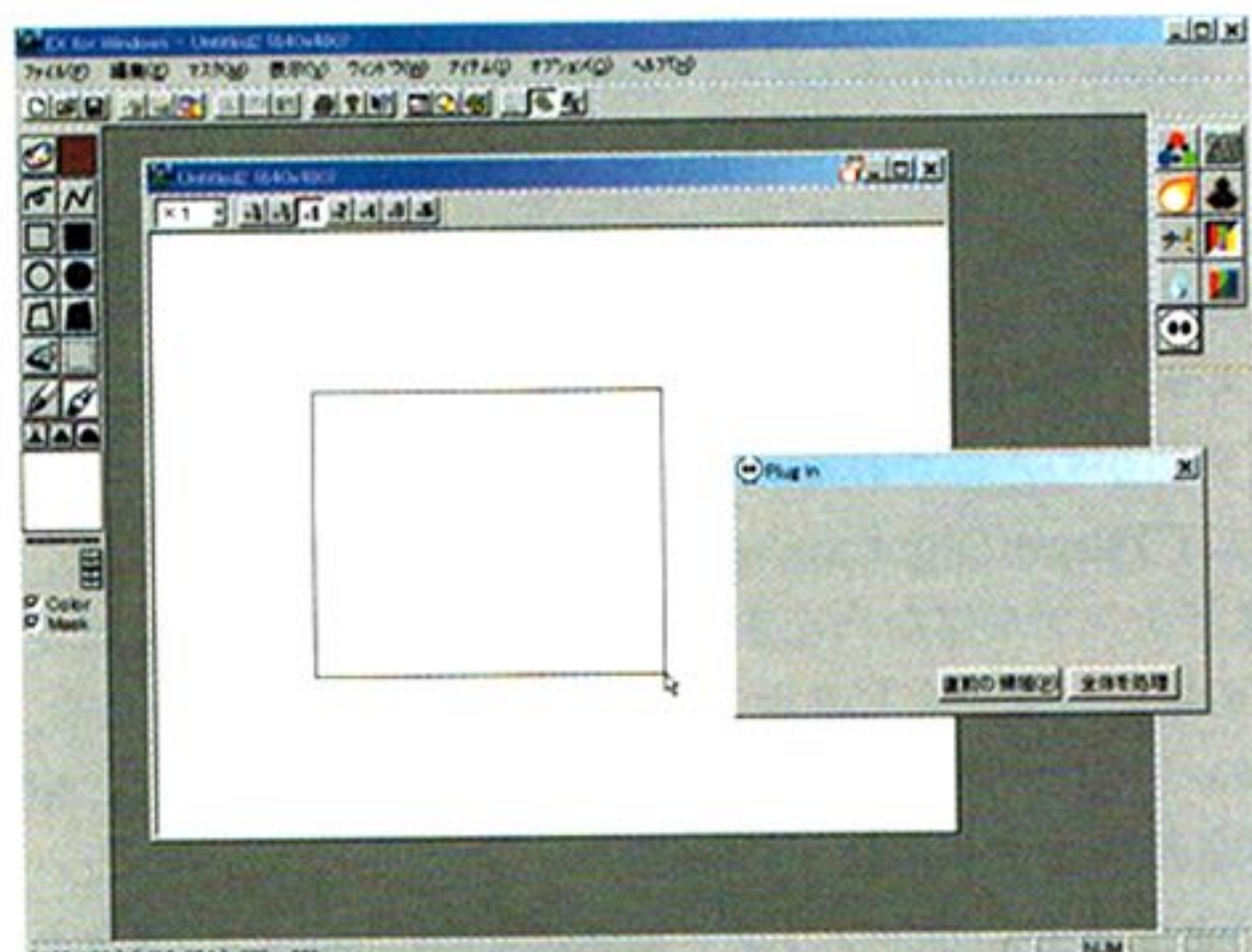



図2 スケルトンをピュアな状態でコンパイルしたもの。すでに領域選択までの部分はできている

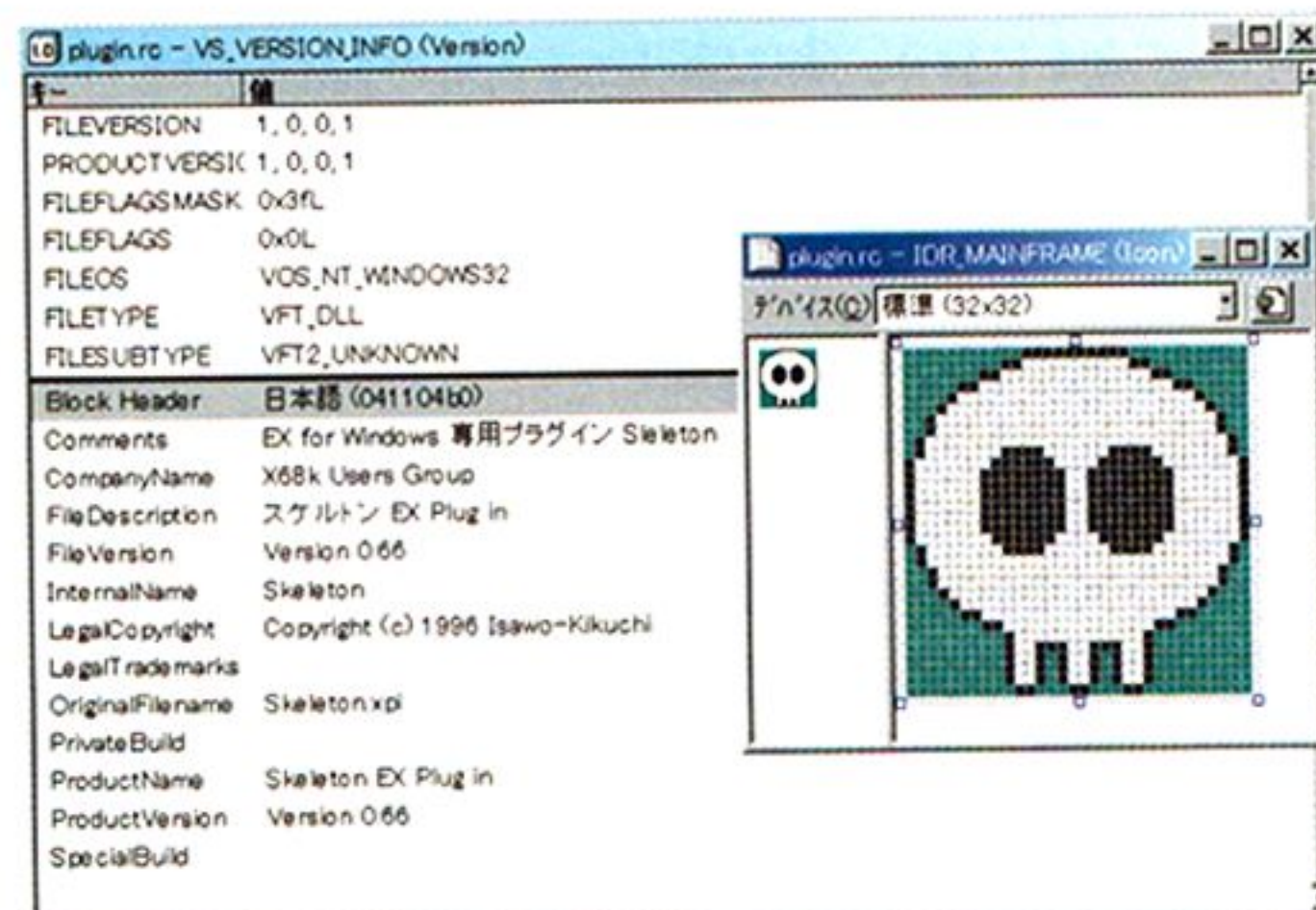



図4 標準のアイコンとバージョンリソース

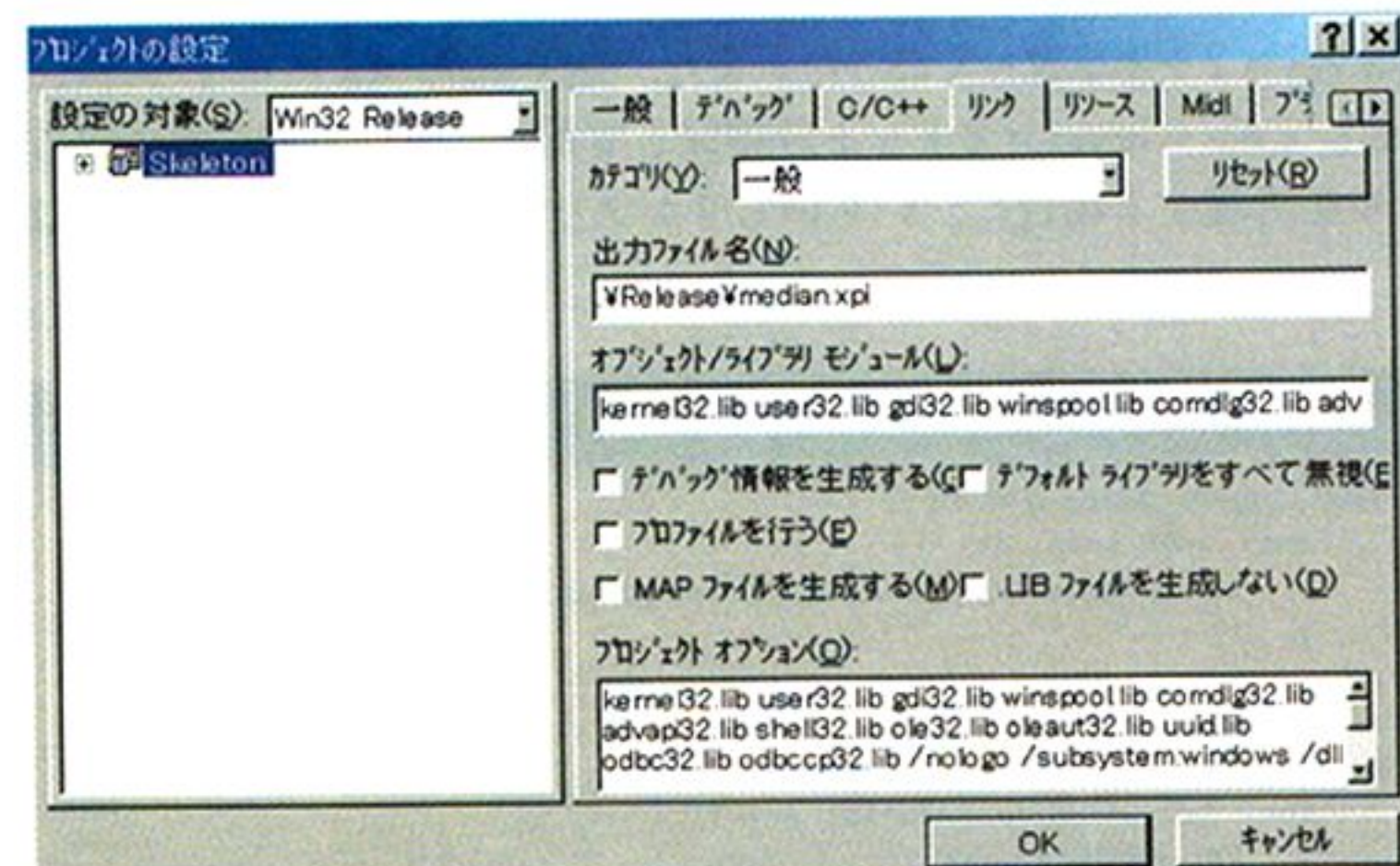



図3 出力ファイル名を変更する

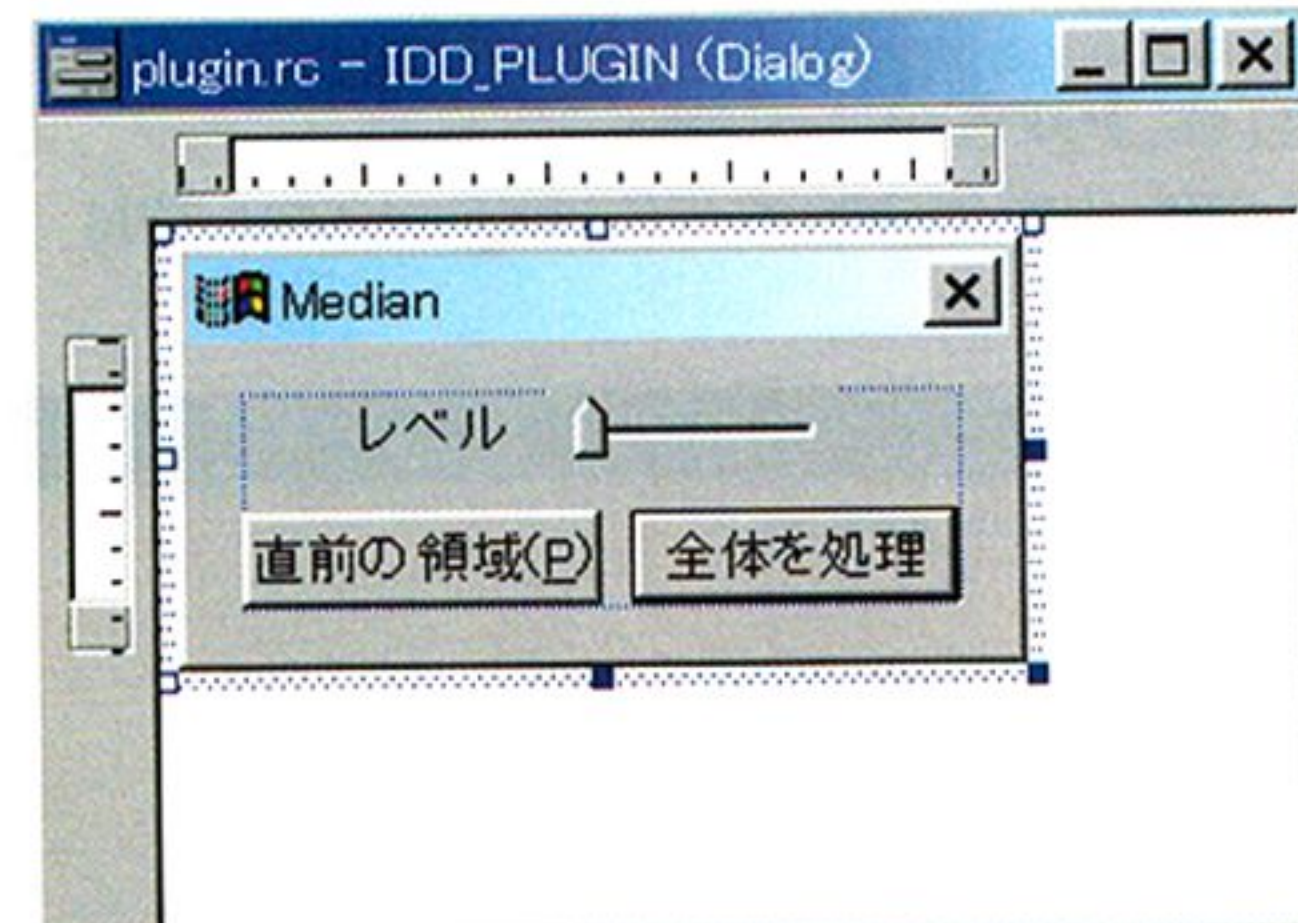



図5 ダイアログにレベルを指定するスライダをつける

プラグインの開発環境

EX用のプラグインとしては，現在次のようなものが用意されている。

パターンブラシ　　フレア
ぼかし　　フラクタル
リバース　　エンボス
サイクル　　ガンマ補正

フィルタ系が多いが，ドロー系の作成も可能だ。というか，その辺もプログラム次第なんである。これらのプラグインの拡張子はxpiとなっているが，中身はDLLである。であるから，それなりの開発環境が必要となる。Visual C++（4.0以上で大丈夫なはず）用のスケルトンを用意してあるが，それ以外の場合は各自でなんとかしてもらいたい。

とりあえず，リファレンスとしてAPI関数などをざっと書き出してみた。ずいぶん前のことなので，自分でもなんでそんな仕様にしたのかよくわからなかったりする部分もある。関数名もそのときの気分で適当につけているので，あまり気にしないでもらいたい。ざっと目を通してもらって，だいたいどんな関数が用意されているのかだけを大雑把に把握しておくといいだろう。システムからコールされる関数については，スケルトンがハンドルして，必要なものは適当に処理してあるので，あまり気にしなくてよい。ちょっと解説が必要なものだけ取り上げてみようか。

EXAPI unsigned char *EXAPILockDocDIB(eDIBType);

eDIBType列挙型で指定したDIBのビット列を取得できる。DIBの内部構造については，VisualC++のほうの記事を参照のこと。ヘッダは含まれていない。eDIBType_Colはカラー部分のDIB，eDIBType_Maskはマスク部分DIBである。eDIBType_Mixはカラーとマスクが合成された，表示時に使われるDIBだ。マスクDIBは8ビットカラー，それ以外は24ビットカラー固定である。eDIBType_UndoCol，eDIBType_UndoMaskはそれぞれアンドゥーバッファのカラー，マスクDIBである。確保されていないDIBを取得しようとすると，NULLが返る。

EXAPI void EXAPICoverMask(RECT);

カラーパートとマスクパートを合成し，ミックスパートを更新する。ミックスパートを自前で更新するのでなければ，これをしないとカラーパートとマスクパートの描画結果が画面に反映されない。基本はEXAPILockDocDIB()で取得したカラーパートとマスクパートをエディットし，そのあとでEXAPICoverMask()でミックスパートを更新，EXAPIShowRect()で画面更新になる。

なお，APIおよびプラグインが受け取る座標はすべてドキュメント座標，つまり画像の座標である。ビュー座標との変換はシステムがやってくれるので，ビューのスクロール位置や表示スケールを気にする必要はない。

簡単なフィルタを作る

まずはスケルトンのプロジェクトを開いてみよう。その状態で，とにかくまずビルドしてみる。そうしてできたSkeleton.xpiをEXにロードし（プラグインはEX.exeと同じフォルダにコピーすること），実行してみると，ダイアログが出るはずだ（図2）。その状態で画像を作ってビュー内でクリックすると，その点とマウスカーソルを対角とする矩形が表示される。マウスを移動させて，適当なところでクリックすると，その矩形は消えてしまうが，もうなんとなくわかっただろう。クリックした2点を対角とする矩形にフィルタをかけるだけならば，あとはフィルタ処理を書くだけで完成なのだ。具体的にいうと，CPluginDlgクラス（plugin.cpp）内の，

Execute(POINT p1, POINT p2)

の内容を書くだけだ。引数はもちろん矩形の対角を示している。ちなみにダイアログに載っている［直前の領域］ボタンは以前に処理した矩形座標を，［全体を処理］ボタンは画像全体の矩形座標を渡してExecute()関数を呼ぶようになっている。

簡単な例として，メディアンフィルタでも作ってみよう（サンプルMedian）。メディアンフィルタとは平滑化フィルタの一種である。ターゲットピクセルの近傍の色をサンプリングし，それらを明度順に並べたときの真ん中の色をターゲットピクセルにすることからその名がついている。まずは名前の変更だ。［プロジェクト］-［設定］を開き，［リンク］タブの［出力ファイル名］の“Skeleton.xpi”を“median.xpi”に変更，これを左上の［設定の対象］が

“Release”，“Debug”ともに行う。次にSkeleton.cppのEntryDLL()関数内の，

pEXInfo->pProfileName = "Skeleton"；

を，

pEXInfo->pProfileName = "Median"；

とする。この文字列は，レジストリキーに使われる。必要ならばアイコンやバージョンリソースも変更するとよい（図4）。

さて，メディアンフィルタの特性上，あまり意味がないかもしれないが，サンプリングする領域の広さも指定できるようにしてみよう。1から3のレベルをスライダで指定できるようにし，それぞれターゲットピクセルを中心とした3×3，5×5，7×7近傍をサンプリング領域とする。図5のようにダイアログをデザインし，plugin.cppに次を付け加えよう。

#include "commctrl.h"

CPluginDlg：：OnInitDialog()内

SendDlgItemMessage(m_hWnd, IDC_LEVELSLIDER, TBM_SETRANGE, FALSE, MAKELONG(1,3))；

SendDlgItemMessage(m_hWnd, IDC_LEVELSLIDER, TBM_SETPOS, TRUE, EXAPIGetProfileInt("Level", 1))；

CPluginDlg：：PostNcDestroy()内

EXAPIWriteProfileInt("Level", SendDlgItemMessage(m_hWnd, IDC_LEVELSLIDER, TBM_GETPOS, 0, 0))；

こうしておけば，スライダの値は終了時にレジストリに保存され，次回起動時に前回のスライダ値が引き継がれる。

そうしたら，最後にCPluginDlg::Execute()関数を仕上げよう（リスト1）。なんか思ったほどシンプルにならなかった。この手の周囲のピクセルを参照するフィルタの場合，次々とピクセルが描き換えられてしまうため，たいていは数ラインのバッファが必要になる。RGBのバッファとともに，中間値の判定に使う明度のバッファも取ってある。

スライダ値をlevelとすると，サンプリング領域の1辺はlevel×2＋1となる。ラインバッファはlevel＋1本にして，ターゲットピクセルよりも上をラインバッファから，下をDIBから参照するというのも可能なのだが，面倒なのでlevel×2＋1本取っておいた。また，処理領域の幅をdxとすると，1ラインとしてdx＋level×2ドット分確保してある。処理領域の端のほうのピクセルを処理する場合に，いちいちサンプリング領域が処理領域をは

リスト1

```
void CPluginDlg::Execute( POINT p1, POINT p2 )
{
   // カラーDIBの取得
   unsigned char *col = EXAPILockDocDIB( eDIBType_Col );
   if( col==NULL ) return;
   // マスクDIBの取得
   unsigned char *mask = EXAPILockDocDIB( eDIBType_Mask );
   // ウェイトカーソル
   SetCursor( LoadCursor( NULL, IDC_WAIT ) );
   // スライダからレベル値を取得
   int level = SendDlgItemMessage( m_hWnd, IDC_LEVELSLIDER, TBM_GETPOS, 0, 0 );
   // 画像のサイズ
   SIZE size;
   EXAPIGetDocSize( size );
   // 座標を整える
   if( p1.x>p2.x ){ int tmp=p1.x; p1.x = p2.x; p2.x = tmp; }
   if( p1.y>p2.y ){ int tmp=p1.y; p1.y = p2.y; p2.y = tmp; }
   // カラーDIBの1ラインバイト数（DWORD境界）
   int lbCol = (size.cx*3+3)/4*4;
   // マスクDIBの1ラインバイト数
   int lbMask = (size.cx+3)/4*4;
   // ラインバッファ
   // 近傍のRGB値を保持するポインタ
   RGBTRIPLE **rgbbuf = new RGBTRIPLE *[level*2+1];
   // 近傍の明度を保持するポインタ
   short **lbuf = new short *[level*2+1];
   int dx = p2.x-p1.x+1;
   rgbbuf[0] = (RGBTRIPLE *)GlobalAlloc( GPTR, (dx+level*2)*(level*2+1)*(sizeof
(RGBTRIPLE)+sizeof(short)) );
   for( int i=1; i<level*2+1; i++ ) rgbbuf[i] = &rgbbuf[i-1][dx+level*2];
   lbuf[0] = (short*)&rgbbuf[i-1][dx+level*2];
   for( i=1; i<level*2+1; i++ ) lbuf[i] = &lbuf[i-1][dx+level*2];
   // カラービット列の初期位置
   col += p1.x*3+(size.cy-p1.y-1)*lbCol;
   // マスクビット列の初期位置
   if( mask ) mask += p1.x+(size.cy-p1.y-1)*lbMask;
   // ラインバッファ取得用のDIBポインタ
   unsigned char *col2 = col;
   // バッファの初期化
   // 明度が-1は範囲外（計算に入れない）
   memset( (void*)lbuf[0], -1, (dx+level*2)*(level*2+1)*sizeof(short) );
   for( int bln=level; bln<level*2; bln++ ){
      GetBuffer( col2, dx, &rgbbuf[bln][level], &lbuf[bln][level] );
      col2 -= lbCol; // 次ライン
   }
   int ln = level*2+1;
   // 明度が真中のピクセルを探すためのテーブル
   short *table = new short[ln*ln];
   // カラー改ライン時の減算バイト数
   int lfCol = lbCol + dx*3;
   // マスク改ライン時の減算バイト数
   int lfMask = lbMask + dx;
   // プログレスバー
   EXAPICreateProgressBar( "ﾒﾃﾞｨｱﾝﾌｨﾙﾀ" );
   int x, y, j, n, max;
   for( y=p1.y; y<=p2.y; y++, bln=(bln+1)%ln ){
      // バッファへ1ライン取得
      if( y+level<=p2.y ){
         GetBuffer( col2, dx, &rgbbuf[bln][level], &lbuf[bln][level] );
         col2 -= lbCol;
      } else for( i=level; i<dx+level; i++ ) lbuf[bln][i] = -1;
      for( x=p1.x; x<=p2.x; x++ ){
         // 明度のテーブルへのセットと有効画素のカウント
         for( j=n=0; j<ln; j++ ){
            for( i=0; i<ln; i++ ){
               if( (table[i+j*ln]=lbuf[j][i+x-p1.x])>=0 ) n++;
            }
         }
         // 真中のピクセルの検索
         for( n/=2; n>=0; n-- ){
            for( i=0, max=-1; i<ln*ln; i++ ){
               if( table[i]>max ){
                  max = table[i];
                  j = i;
               }
            }
            table[j] = -1;
         }
         // ピクセルのセット
         if( mask ){// マスクバッファあり
            *(col++)      +=      ((rgbbuf[j/ln][(j%ln)+x-p1.x].rgbtBlue-
(*col))*(*mask)+128)>>8;
            *(col++)      +=      ((rgbbuf[j/ln][(j%ln)+x-p1.x].rgbtGreen-
(*col))*(*mask)+128)>>8;
            *(col++)      +=      ((rgbbuf[j/ln][(j%ln)+x-p1.x].rgbtRed-
(*col))*(*mask)+128)>>8;
            mask++;
         } else {   // マスクバッファなし
            *((RGBTRIPLE*)col) = rgbbuf[j/ln][(j%ln)+x-p1.x];
            col += sizeof(RGBTRIPLE);
         }
      }
      col -= lfCol;
      if( mask ) mask -= lfMask;
      EXAPISetProgressBarValue( (y-p1.y)*100/(p2.y-p1.y) );
   }
   EXAPIDeleteProgressBar();
   delete []table;
   GlobalFree( (HGLOBAL)rgbbuf[0] );
   delete []rgbbuf;
   delete []lbuf;
   // DIBを開放
   EXAPIUnlockDocDIB( eDIBType_Col );
   EXAPIUnlockDocDIB( eDIBType_Mask );
   // ミックスパートの更新
   if( rect.left<0 ) rect.left = 0;
   if( rect.top<0 ) rect.top = 0;
   if( rect.right>size.cx ) rect.right = size.cx;
   if( rect.bottom>size.cy ) rect.bottom = size.cy;
   RECT rect = { p1.x, p1.y, p2.x+1, p2.y+1 };
   EXAPICoverMask( rect );
   EXAPIShowRect( rect );
   EXAPISetModifiedFlag( TRUE );
   // 矢印カーソル
   SetCursor( LoadCursor( NULL, IDC_ARROW ) );
}

void CPluginDlg::GetBuffer( unsigned char *dib, int width, RGBTRIPLE *rgb,
short *l )
{
   memcpy( (void*)rgb, (void*)dib, width*sizeof(RGBTRIPLE) );
   for( int i=0; i<width; i++ ){
      // RGBに適当に比重をつける
      l[i] = (rgb[i].rgbtRed<<1)+(rgb[i].rgbtGreen<<2)+rgb[i].rgbtBlue;
   }
}
```

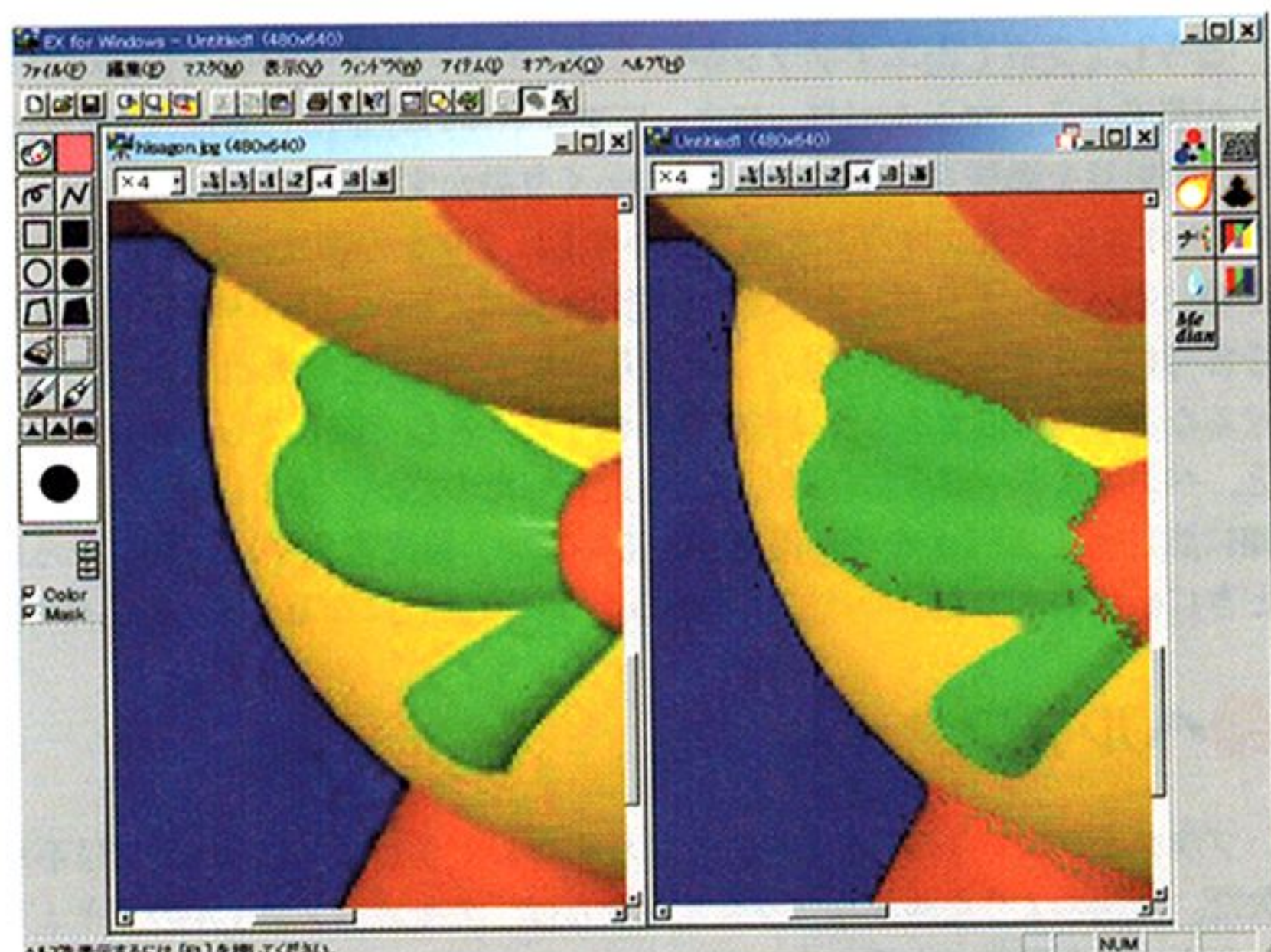



図6　レベル3でメディアンフィルタをかけた例（右）。ノイズは減っているが，派手に輪郭が壊れている

み出していないか判定する手間を省くためだ。それぞれのポイントが有効（処理領域内）であるかどうかは，明度が正であるかどうかで判定している。さらに，中間値の検索のために，サンプリング領域と同じサイズの配列が必要となる。

こういったアルゴリズムはフィルタによってさまざまなので，あまり本質的ではない。重要なのは点の描き込み方法である。もしマスクが確保されていなければ（EXAPILockDocDIB（eDIBType_Mask）でNULLが返れば），EXAPILockDocDIB（eDIBType_Col）で取得したDIBに対して描き込んでいけばいいだけなのだが，マスクがあった場合は，マスクの透明度を考慮しなければならない。下地の色がc0としよう。ここにはマスクmが載っており，その上からc1を描き込むとする。その場合，描き込んだあとの色c0'とすると，

$$c0' = \frac{c0 \times (255-m)}{255} + \frac{c1 \times m}{255}$$

となる（mは0で不透明，255で透明）。これを少し変形すると，

$$c0' = \frac{c0 + (c1-c0) \times m}{255}$$

となるのだが，これは1ピクセルごとに処理される非常に頻度の高い式である。そこで，「255で割る」という重い処理を「8ビットシフトダウンする（256で割る）」にしてしまうと，かなりの高速化が図れるわけだ（昨今のCPUでは変わらないかもしれないのだが，たぶん遅くなることはないはず）。つまりC++では次のように書ける。

```
c0 += ((c1-c0) * m + 128) >>8;
```

c0，c1をRGBに対応させて，この式を3回計算することになる（128を加算しているのは，四捨五入のため）。誤差については，人間の目ではわからない程度なので，許容範囲としてもいいだろう。問題があるとすれば，mが255であった場合，つまり完全透明であった場合でも，上から描き込んだ色が正確に反映されないことだが，気になるようならm＝255だけ場合分けするといい。EXのシステム内部は，すべてこの計算式によっている。

結構複雑になってしまった割には，完成したフィルタはあまり質が高くならなかった。というか，やはりレベルは最小にしないと，輪郭がかなり派手に壊れて実用にならない（図6）。サンプリング領域を矩形にしたのもまずかったのだろう。あと，多分大丈夫だと思うのだが，ひょっとしたらどこか間違っているかもしれない。これでも割と見やすさを優先したので，そんなに速くもないし。ま，プラグイン作成法のサンプルということで勘弁してもらいたい。

ドローイング系

フィルタ系プラグインは簡単に作成できることはおわかりいただけただろう。サンプルとして取り上げたメディアンフィルタは，周囲のサンプリングが必要なためにバッファを取るなど少々ごちゃごちゃしてしまったが，たとえば，明度を調節するなど，ターゲットピクセルで完結しているフィルタならば，C++は初めてという人でも作成できるのではないだろうか。それ以上のことに関しては，調べるなり精進してもらうとして，ドローイング系プラグインについても手を出しておこう。

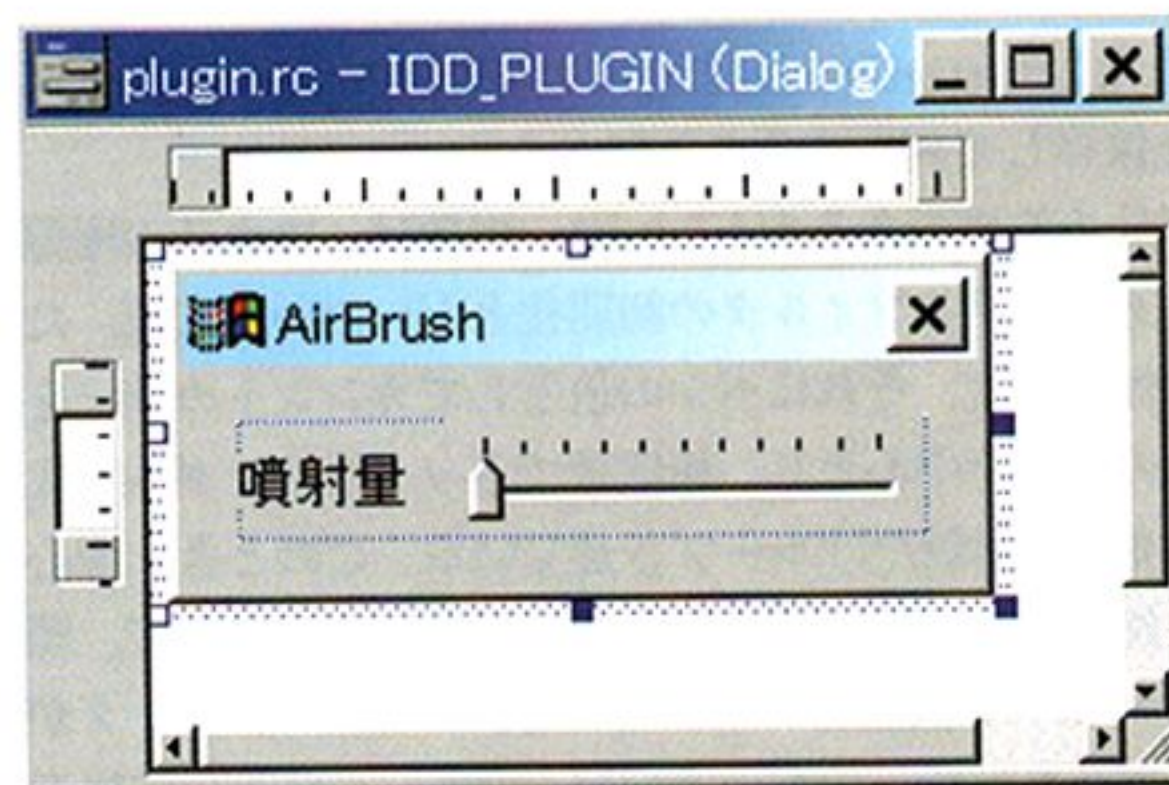



図7　ボタンを削除し，噴射量のスライダをつける

こちらはできるだけ簡単なものをということで，エアブラシを作ることにした。マウスの左ボタンを押している間，カーソルを中心とした円内の中心に比重を置いて，ランダムにドットを噴きつけるというものだ（サンプルAirBrush）。噴きつける領域はシステム側のペンサイズを利用し，その濃度もシステムのペン濃度を使用することにする。

まずはSkeletonをコピーしたら，先ほどと同じように出力ファイル名，レジストリで使用されるProfile名，バージョンリソース，アイコンリソースを修正する。ダイアログリソースは，今回は［直前の領域］［全体を処理］が必要ないので削除し，代わりにまたスライダを置く。このスライダはドットの噴射量である。［直前の領域］がなくなったことで，レジストリや関連を削除し，先ほどと同様にスライダの値のレジストリ操作を追加する。さて，ここからだ。さっきは左クリックしたときの処理はスケルトンソースに任せていたが，今度は自分で処理しなければならない。必要な処理は次のような感じだろう。

・LButtonDown
マウスのキャプチャを行う。また，EXAPIEnableCursorLeave（）はFALSEにしよう。この関数は，説明だけではなにをいいたいのかよくわからないだろうが，こういうことだ。マウスをキャプチャした場合，マウスがビューの外に出ると自動スクロールする。EXAPIEnableCursorLeave（）がTRUEの場合（デフォルト）は，マウスを再びビューの中に戻すか，スクロールが終端に達するまで，スクロールしっぱなしとなる。それに対してFALSEにすると，いったんスクロールしたあとにマウスを強制的にビューの中に戻すので，マウスを動かした分しかスクロールしない。概してTRUEは領域指定の場合など，FALSEは描画時などに指定するとよい。

・LButtonUp
マウスのキャプチャをリリースする。

・MouseMove
もしマウスがキャプチャされていれば，その座標にエアブラシで描画を行う。ステータスバーに座標を表示する処理は，スケルトンのものをそのまま利用すればよい。

さて，このままではマウスが動いた瞬間しか描画が行われない。左ボタンを押した瞬間から処理をがめこんでしまい，ボタンが離されるまでぶんぶんループを回すという手もあるが，それではほかのメッセージの処理が滞ってしまうし，だいいちWindowsプログラムのルールに反する。そこで登場するのがアイドル関数Idleである。Idleはメッセージキューにメッセージがないとき，つまり暇なときに呼ばれる関数である。戻り値としてTRUEを返すと，暇なときは何回でも連続して呼び出され，FALSEを返しても，ほかになにかメッセージがあると，そのあとにまたFALSEが返されるまで呼ばれ続ける。つまりこのIdleで，マウスをキャプチャしているときは描画処理を行いTRUEを返し，そうでなければFALSEを返せばよい。このとき，描画を行う座標はEXAPIGetDocPoint（）で取得してもいいのだが，どうせ

座標は直前に送られてきたLButtonDownまたはMouseMoveの座標と同じはずなので，そちらを保存して利用することにした。

というわけで，描画部分の処理である（リスト2）。引数は，マウスの座標である。最初のほうは，メディアンフィルタの初期化とだいたい同じだ。ただし，こちらはペンの半径と濃度，それにペンの色としてカレントカラーを取得している。カーソルの座標を中心とし，極座標でドットをランダムで生成，それを噴射量（の10倍）の回数だけループさせている。このとき，中心のほうを密にするため，距離はランダム値を2乗して求めている。さて，問題はドットの描画だ。今回はペン濃度とマスクという，2つのパラメータがある。ペン濃度は1から255までで，値が大きいほど濃い（0は透明で無意味）。マスクのほうも，結果的に値が大きいほどペンの色が反映されるわけだから，この2つのパラメータは単純に掛け合わせてしまい，それを濃度とすればよい。濃度を1，マスクをmとすると，次のようになる。

$1' \div 255 = 1 \div 255 \times m \div 255$（÷は分数で）

$1' = 1 \times m \div 255$

ここでまた255で割る処理をビットシフトにしてしまってもよいのだが，エアブラシの性質上，ここはそれほど速い処理が必要なわけではないので，精度を優先してこのままとした。

リスト2

```
void CPluginDlg::Execute( POINT p1 )
{
    // スライダから噴射量を取得
    int count = SendDlgItemMessage( m_hWnd, IDC_COUNT, TBM_GETPOS, 0, 0 )*10;
    // カラーDIBの取得
    unsigned char *col = EXAPILockDocDIB( eDIBType_Col );
    // マスクDIBの取得
    unsigned char *mask = EXAPILockDocDIB( eDIBType_Mask );
    // 画像のサイズ
    SIZE size;
    EXAPIGetDocSize( size );
    // カラーDIBの1ラインバイト数（DWORD境界）
    int lbCol = (size.cx*3+3)/4*4;
    // マスクDIBの1ラインバイト数
    int lbMask = (size.cx+3)/4*4;
    // カラービット列の初期位置
    col += p1.x*3+(size.cy-p1.y-1)*lbCol;
    // マスクビット列の初期位置
    if( mask ) mask += p1.x+(size.cy-p1.y-1)*lbMask;
    // ペンの半径と濃度を取得
    int radius = EXAPIGetPenRadius();
    int level = EXAPIGetPenLevel();
    int red = EXAPIGetCurCol( eCol_Red );
    int green = EXAPIGetCurCol( eCol_Green );
    int blue = EXAPIGetCurCol( eCol_Blue );
    int r, t, x, y, ml;
    const double pi = 3.14159265358979;
    double d;
    unsigned char *c, *m;
    if( !mask ) ml = level;
    for( int i=0; i<count; i++ ){
        r = (((radius*rand())>>15)*((radius*rand())>>15))/radius;
        t = (360*rand())>>15;
        d = r*sin( t*pi/180 );
        x = (int)((d>0.0)?(d+0.5):(d-0.5));
        d = r*cos( t*pi/180 );
        y = (int)((d>0.0)?(d+0.5):(d-0.5));
        if( p1.x+x>=0 && p1.x+x<size.cx ) if( p1.y+y>=0 && p1.y+y<size.cy ){
            c = col+x*3-y*lbCol;
            if( mask ){ // マスクバッファあり
                m = mask+x-y*lbMask;
                ml = ((*m)*level)/255;
            }
            *(c++) += ((blue-(*c))*(ml)+128)>>8;
            *(c++) += ((green-(*c))*(ml)+128)>>8;
            *(c++) += ((red-(*c))*(ml)+128)>>8;
        }
    }
    // DIBを開放
    EXAPIUnlockDocDIB( eDIBType_Col );
    EXAPIUnlockDocDIB( eDIBType_Mask );
    // ミックスパートの更新
    RECT rect = { p1.x-radius, p1.y-radius, p1.x+radius+1, p1.y+radius+1 };
    if( rect.left<0 ) rect.left = 0;
    if( rect.top<0 ) rect.top = 0;
    if( rect.right>size.cx ) rect.right = size.cx;
    if( rect.bottom>size.cy ) rect.bottom = size.cy;
    EXAPICoverMask( rect );
    EXAPIShowRect( rect );
    EXAPISetModifiedFlag( TRUE );
}
```

こうして完成したエアブラシが図8だ。ちょっとざらついた質感のブラシが得られる。マシンパワーによってアイドルの呼び出し頻度は変わるので，噴射量を調整するなりマウスをゆっくり動かすなりして感じをつかんでもらいたい。また，マウスを速く動かすと，ブラシの密度が線にならずに，点線になってしまうだろう。これを避けるには，Visual C++のほうの記事を参考にしてラインで処理し，1点辺りのループ回数を移動距離に反比例させるなどして対処すると，綺麗になるはずだ。各自で試してもらいたい。あと，ペンの太さや濃度は保存しておきたかったのだが，保存してもシステム側に設定するAPIがなぜか用意されていない。後先考えずに必要になったときに必要なものだけつけていくからこういうことになる。困ったものだ。

ヘルプファイル

プラグインはできたとしても，やはり配布しようと思ったらヘルプは不可欠だ。今回のような簡単なプラグインならば，テキストファイルのドキュメントをつける程度でも構わないだろうが，EX側にはプラグインのヘルプをサポートするAPIも用意されている。その辺りはスケルトンで処理してあるので気にする必要はないが，ヘルプは各自で作成しなければならない。最近のヘルプファイルの流行はchm（Compressed HTMLの略，前号のDirect3D Retained Mode講座で書いたCompiled HTMLは間違い）であるが，EXのAPIを作った頃はそんなものはまだ影も形もなく，オンラインヘルプといえばWindowsヘルプ（HLPファイル）を指した。そんなわけで，HLPの作成もプラグイン作成の一環となる。

私はHELPファイルの作成には，梅木泰宏氏作成のヘルプカード（シェアウェア2500円，URL:http://www2b.biglobe.ne.jp/~mono/）を使わせてもらっている。リッチテキストでコンテンツを書いて，Visual C++なりVisual Basicに付属のHelp Workshopでヘルプは作成できるのだが（ヘルプカードもHelp Workshopを呼び出している），それなりに面倒なので，こういったものを利用したほうが本質的でない部分で悩む必要がなくてよいだろう。初期の設定は必要だが，ちょちょっとコンテンツを書いて，ボタンをポンと押せば，ヘルプファイルのできあがりである。

ヘルプファイル名は，プラグインのベース名に合わせておけば，EXが自動的に判別してくれる。このヘルプファイルが呼ばれるのは，プラグインのダイアログにフォーカスがあるときF1キーを押した場合，およびツールバーの［状況依存のヘルプ］ボタン（矢印カーソルにクエスチョンマークがついたボタン）を押し，プラグインのダイアログ内をクリックした場合である。このヘルプファイルはプラグインと同じフォルダ，つまりEXの本体と同じフォルダに置いておく。

最後に

EXはまだ制作途中であることは先に述べたとおり。今回サンプルのプラ

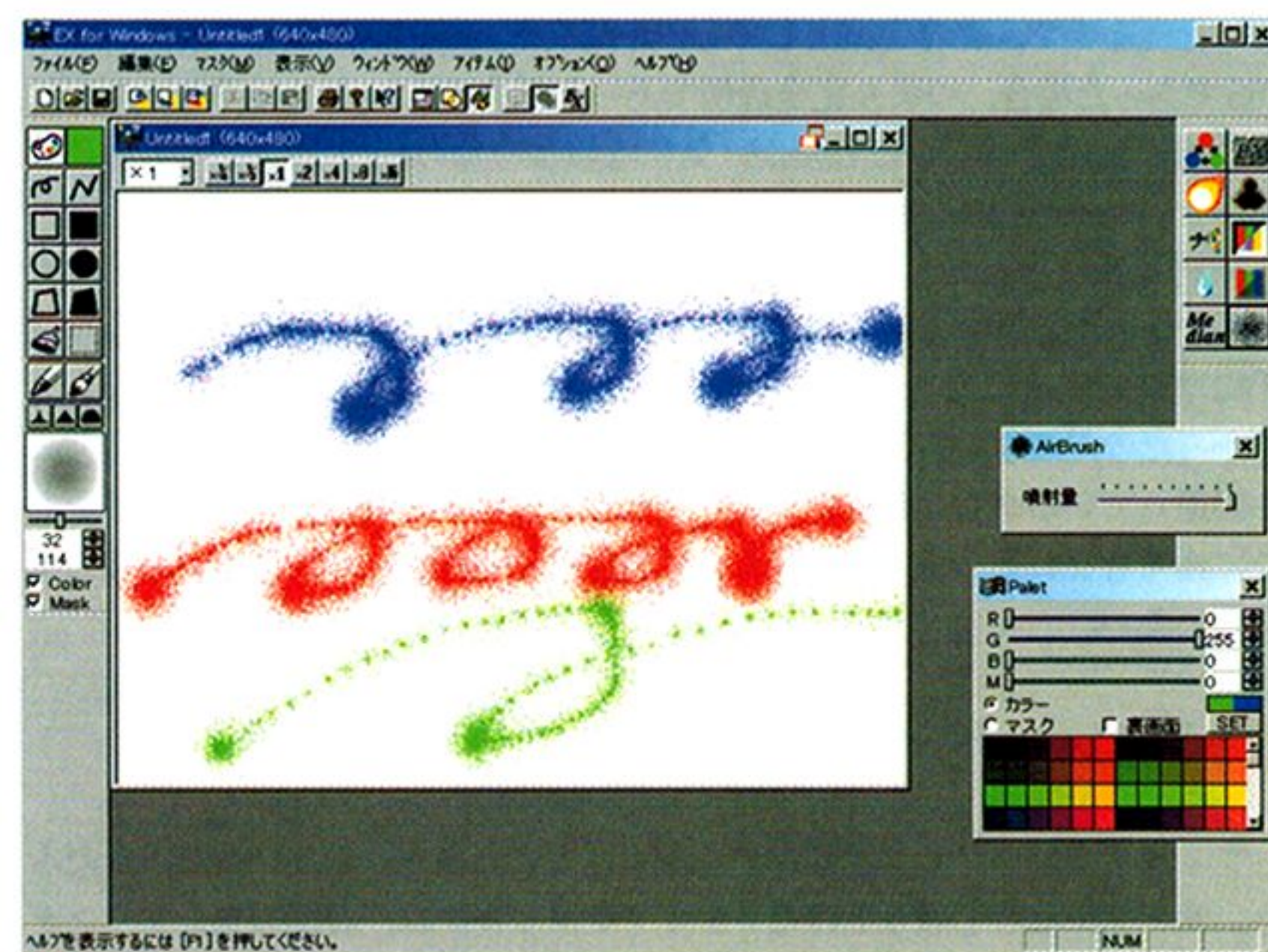



図8 AirBrush

グインを作るにあたって,「こういうAPIがあったら」とか「なんでこのAPIがない」というのを,作った本人でありながらかなり感じた。現状ではハマっている部分を打開しないことには,その辺りまで手を回すことができない状態なのだが,とりあえずいまあるAPIの仕様変更はないはずだ。まあ,なんとかなったら,またこの誌上で今度は本体の紹介もちゃんと含めてできるかもしれないので,なにかプラグインができたら,Oh!Xまで投稿してみてもらいたい。

EX for Win APIリファレンス　Column

列挙型

カラー要素

```
typedef enum eCol {
        eCol_Red,
        eCol_Green,
        eCol_Blue,
        eCol_Mask,
} eCol;
```

DIBメモリブロックタイプ

```
typedef enum eDIBType {
        eDIBType_Col,
        eDIBType_Mask,
        eDIBType_Mix,
        eDIBType_UndoCol,
        eDIBType_UndoMask,
} eDIBType;
```

構造体

システムから渡されるインフォメーション

```
typedef struct EXAPIInfo {
        HWND hParentWnd;
        WPARAM      wParam;
        LPARAM      lParam;
        RGBTRIPLE *pMaskPal;
        char *pProfileName;
} EXAPIInfo;
```

API関数

EXAPI void EXAPIPostExitMessage (void);
　プラグインが終了することをシステムに知らせる。

EXAPI unsigned char * EXAPILockDocDIB (eDIBType);
　カレントドキュメントの指定のDIBをロックし,ビット列のメモリアドレスを取得する。

EXAPI void EXAPIUnlockDocDIB (eDIBType);
　EXAPILockDocDIB()で取得したDIBをアンロックし,メモリを解放する。

EXAPI void EXAPIGetDocSize (SIZE &);
　カレントドキュメントのサイズを取得する。

EXAPI int EXAPIGetCurCol (eCol);
　カレントカラーの指定要素を取得する。

EXAPI void EXAPIRevRect (RECT);
　反転色で矩形のフレームを表示する。

EXAPI void EXAPICoverMask (RECT);
　カラーパートとマスクパートからミックスパートを更新する。

EXAPI void EXAPIShowRect (RECT);
　矩形領域の表示を更新する。

EXAPI int EXAPIGetPenRadius (void);
　ペンの半径を取得する。

EXAPI int EXAPIGetPenLevel (void);
　ペンの濃度を取得する。

EXAPI void EXAPICapture (BOOL);
　マウスのキャプチャを設定する。引数がTRUEの場合はキャプチャをセット,FALSEの場合はリリース。キャプチャ中はビューはオートスクロールする。

EXAPI void EXAPIGetDocPoint (POINT &);
　現在のカーソル座標をドキュメント座標で取得する。

EXAPI void EXAPISetModifiedFlag (BOOL);
　ドキュメントのModifiedFlagを設定する。

EXAPI void EXAPIGetColor (POINT);
　指定座標のカラーをカレントカラーに設定する。

EXAPI void EXAPIRevPoint (POINT);
　指定座標に反転色で点を表示する。

EXAPI void EXAPIRevLine (POINT, POINT);
　指定座標に反転色でラインを表示する。

EXAPI void EXAPIRevBox (POINT, POINT);
　指定座標に反転色で矩形を表示する。

EXAPI void EXAPIRevEllipse (POINT, POINT);
　第1引数を中心とし,第2引数までの水平変移をX半径,垂直変移をY半径として楕円を表示する。

EXAPI void EXAPIShowStatus (LPCSTR);
　ステータスバーに文字列を表示する。

EXAPI void EXAPIWinHelp (DWORD);
　ヘルプの指定コンテキストを表示。

EXAPI void EXAPICreateProgressBar (LPCTSTR);
　ラベルを指定して,ステータスバーにプログレスバーを作成。

EXAPI void EXAPIDeleteProgressBar (void);
　EXAPICreateProgressBar()で作成したプログレスバーを廃棄。

EXAPI void EXAPISetProgressBarValue (int);
　EXAPICreateProgressBar()で作成したプログレスバーに値(0〜100)を設定。

EXAPI void EXAPIEnablePen (BOOL, BOOL);
　第1引数でペンの太さ,第2引数でペン濃度の有効/無効を設定する。プラグイン起動時のデフォルトはどちらも無効。

EXAPI void EXAPIEnableCursorLeave (BOOL);
　キャプチャ中にマウスカーソルがビューの外に出た場合,スクロールし続けるか(TRUE)いったんスクロールしたあとにカーソルをビュー内に戻すか(FALSE)を指定する。

EXAPI void EXAPIGetProfileString (LPCTSTR, LPCTSTR, LPTSTR, DWORD);
　レジストリから文字列を取得する。

EXAPI UINT EXAPIGetProfileInt (LPCTSTR, int);
　レジストリから数値を取得する。

EXAPI BOOL EXAPIWriteProfileString (LPCTSTR, LPCTSTR);
　レジストリに文字列を格納する。

EXAPI BOOL EXAPIWriteProfileInt (LPCTSTR, int);
　レジストリに数値を格納する。

システムからコールされる関数

extern "C" DllExport BOOL EntryDLL (EXAPIInfo far *);
　プラグインがロードされた直後に呼ばれる。EXAPIInfoのpProfileNameにプラグイン名へのポインタを入れ,初期化を行う。初期化が正常に終了した場合は,TRUEを返す。lParamとwParamはプラグイン側が自由に使える変数。

extern "C" DllExport void ExitDLL (EXAPIInfo far *);
　プラグインが破棄されるときに呼ばれる。終了処理を行う。

extern "C" DllExport BOOL FilterDllMsg (EXAPIInfo far *, LPMSG);
　ダイアログメッセージを処理する必要があるときに呼ばれる。メッセージが処理されたときは,TRUEを返す。

extern "C" DllExport BOOL ProcessDllIdle (EXAPIInfo far *, LONG);
　メッセージアイドル中に呼ばれる。アイドルが必要ない場合はFALSEを返す。

extern "C" DllExport void OnLButtonDown (EXAPIInfo far *, UINT, POINT);
　左ボタンが押されたときに呼ばれる。座標はドキュメント座標で取得される。

extern "C" DllExport void OnLButtonUp (EXAPIInfo far *, UINT, POINT);
　左ボタンが離されたときに呼ばれる。

extern "C" DllExport void OnLButtonDblClick (EXAPIInfo far *, UINT, POINT);
　左ボタンがダブルクリックされたときに呼ばれる。

extern "C" DllExport void OnRButtonDown (EXAPIInfo far *, UINT, POINT);
　右ボタンが押されたときに呼ばれる。

extern "C" DllExport void OnRButtonUp (EXAPIInfo far *, UINT, POINT);
　右ボタンが離されたときに呼ばれる。

extern "C" DllExport void OnRButtonDblClick (EXAPIInfo far *, UINT, POINT);
　右ボタンがダブルクリックされたときに呼ばれる。

extern "C" DllExport void OnMouseMove (EXAPIInfo far *, UINT, POINT);
　マウスが移動したときに呼ばれる。

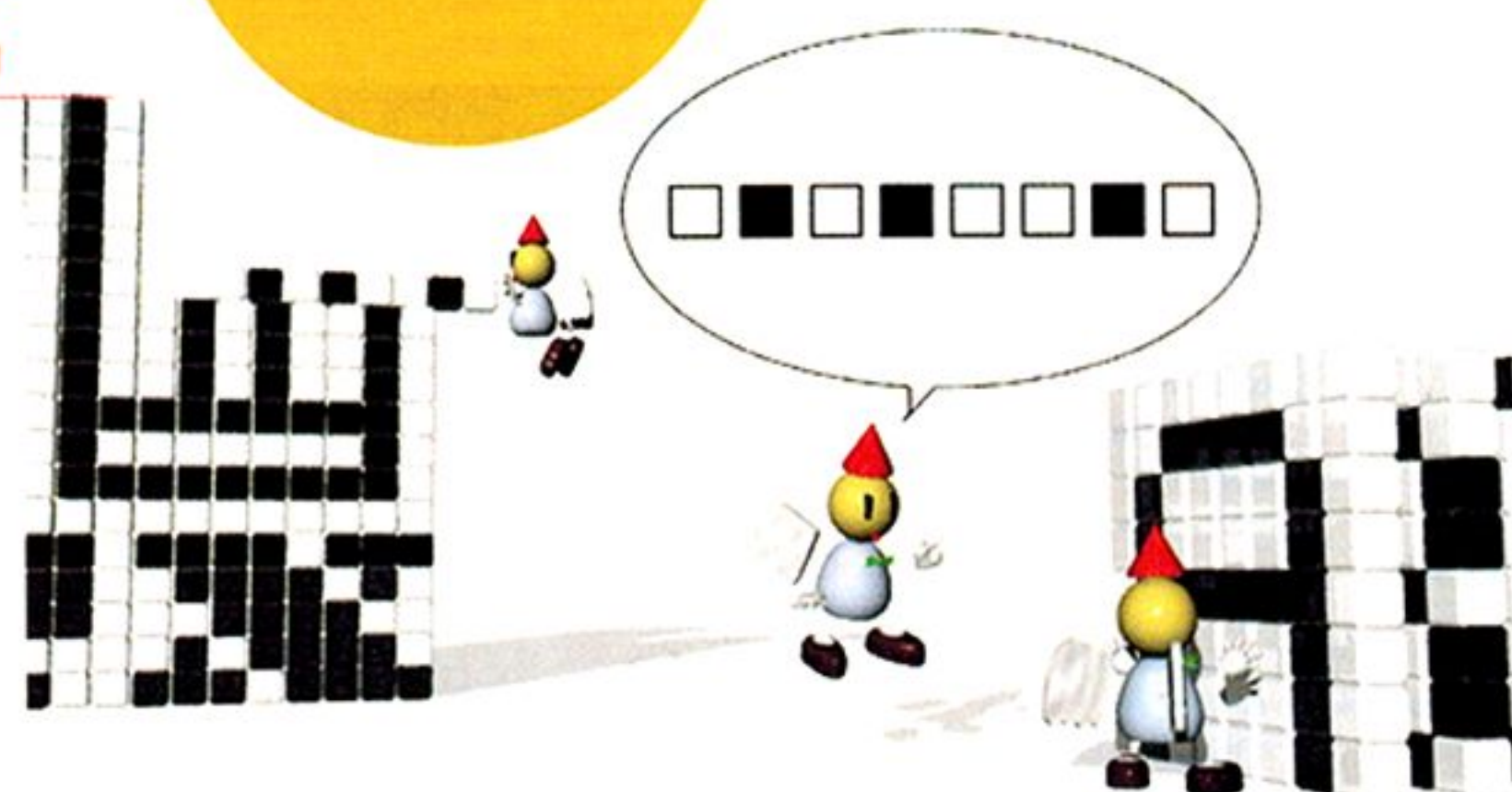
extern "C" DllExport BOOL OnMouseLeave (EXAPIInfo far *);
　マウスがクライアント領域から外に出たときに呼ばれる。デフォルトの処理を行う場合はFALSEを返す。

extern "C" DllExport void OnShowBand (EXAPIInfo far *, BOOL);
　ビューがスクロールした場合や更新する場合など,領域指定中の表示(主に反転表示)を表示/非表示する必要があるときに呼ばれる。

extern "C" DllExport void OnChangedMaskPal (EXAPIInfo far *);
　マスクの表示色が変更されたときに呼ばれる。

無改造MZ-700でビットマップ表示を 番外編

百井 洋 Momoi You

「不可能はない」といわれたマシン，MZ-700の最大の弱点はビットマップグラフィック機能を持っていないことだった。常識的に考えると，文字表示しかできないMZ-700で1ドット単位の画像表現は不可能だ。ここでは無改造のままソフトだけでビットマップ表示を可能にする方法を紹介してみたい。

グラフィックとMZ-700

グラフィックを語るうえで忘れられないパソコンといえばMZ-700です。主にグラフィック画面を持っていないという理由で。知らない人のために一応説明しますと，MZ-700はシャープが1983年に発売したパソコンで，CPUにZ80A (3.57MHz) を搭載し，64Kバイトフル実装のRAMを持っていました。MZ-700のビデオ性能は，

40×25字のテキスト画面
各文字ごとに文字色と背景色を指定可能

というごくシンプルなものでしたが，豊富なキャラクタセットと8色フルカラーを駆使して作り上げる画面の表現力には驚くべきものがあり，本誌でも活躍中の古籏一浩氏によるMZ-700版「tiny XEVIOUS」や「スペースハリアー」はその完成度で人々に大きな衝撃と感動を与えたものです。

とはいえMZ-700ユーザーとしてはやっぱり，キャラグラではない「グラフィック表示」に憧れるじゃありませんか。

ラスタ技

ひと昔前，ゲーム機といえばスーファミやメガドラの頃，ラスタスクロールというものが流行りました。ラスタを1本描くごとにVRAMの表示開始位置をずらすことで，画面がうねうねと波打ったように見えるアレです。

MZ-700はテキスト画面しか持っていませんが，CRTCが上から1ラスタずつ描いていることには変わりありません。CRTCがある場所のキャラクタAの1ラスタ目を描き終わったところで，すかさずキャラクタAをキャラクタBに置き換えてやれば，CRTCが描く2ラスタ目はキャラクタBの2ラスタ目になります。この調子で，たとえば"■"と"(空白)"を1ラスタごとに置き換えてやれば，1キャラクタ分の横縞模様を表示することができます。

こうして，キャラクタをラスタごとに分解したパターンで元絵のビットパターンを近似してやれば，「グラフィック表示」ができそうではありませんか。

グラフィックを分解再構成キャラクタで近似

MZ-700のCGROMデータから，「8ビットのビットパターン」→「該当パターンを第nラスタに持つキャラクタ」の変換テーブルを作ります。といっても対応するパターンがすべてあるわけはなく，むしろないほうが多いので，ないものに関しては，

ハミング距離が最小
"1"のビットの並びがもっとも似ている

という条件を満たすものを選んでいます。ハミング距離とはアバウトにいって，ビットごとに比較したときの異なる箇所の総数です。並びが似ているという判定は次のように行っています。たとえば本来のビットパターンが，

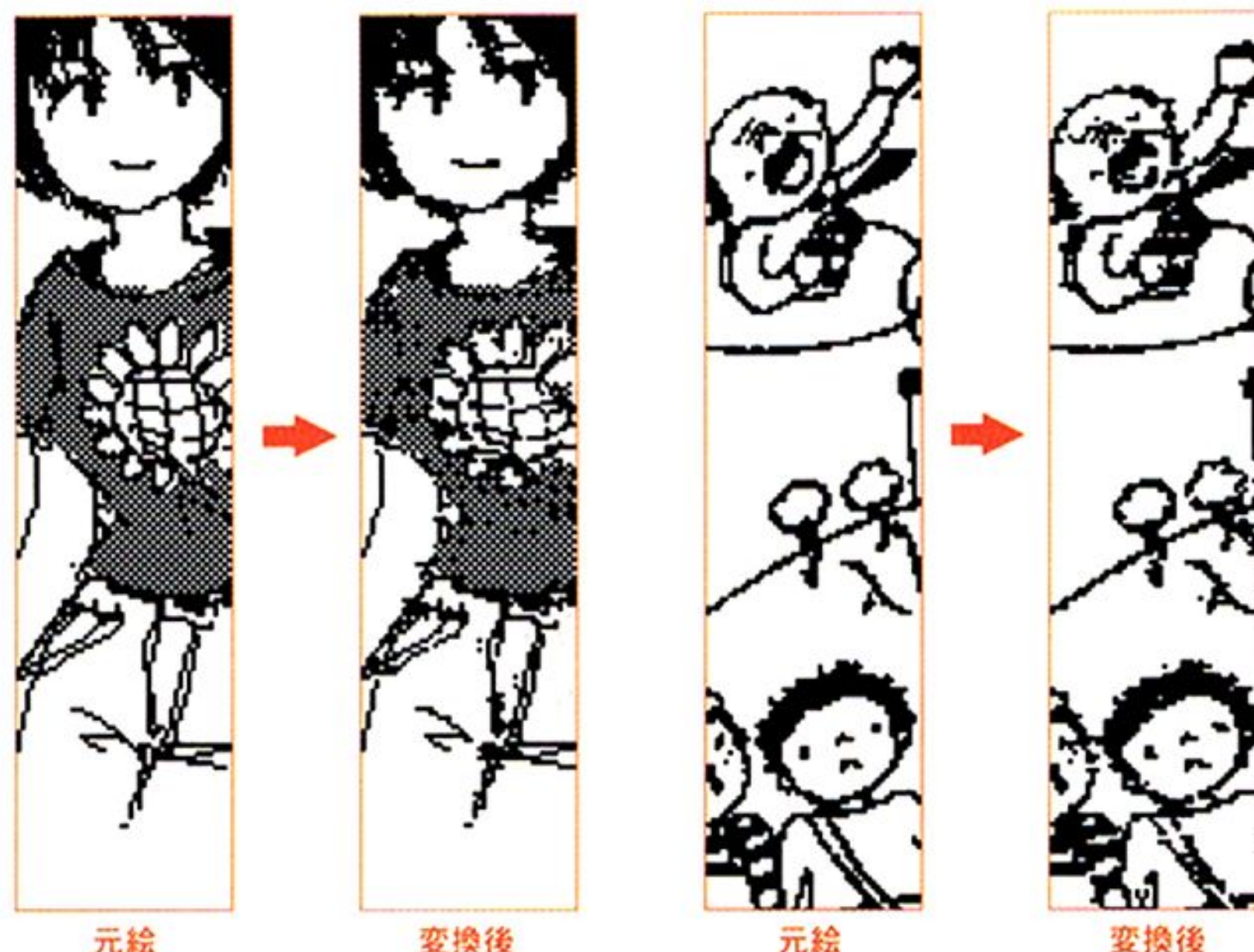
00010011

だったとき，まず各ビットと最寄りの"1"のビットがいくつ離れているかを，

32101100

のように測ります。この値の，判定すべきビットパターンで"1"になっているところを加算し，総和が最小のものを「もっとも似ている」と判定しています。

これが最良の方法というわけではないのですが，適当な割にはそこそこ見られる変換結果が得られたので，とりあえずよしとしています。

元絵　変換後　元絵　変換後

MZ-700の水平表示タイミング

MZ-700ではCPUクロックとCRTCのドットクロックを1個のオシレータ(14.318180MHz)から作っています。CPUクロックは3.579545MHz，ドットクロックは7.159090MHzです。

MZ-700の画面表示における水平表示期間は44.69842μs，水平ブランキング期間は18.99683μsです。ドットクロックから換算して，表示期間320ドット，ブランク期間136ドットです。よって1ラスタは456ドットで，CPUクロックでいえば228クロックで1本のラスタが描かれていることになります。

Z80でもっとも高速なソフトウェア転送はいわゆる「POP-LD転送」と呼ばれるもので，スタックポインタSPに転送元アドレスをセットし，

```
POP   HL        (10)
LD    (nn),HL   (16+2?)
```

をひたすら並べることで実現します。カッコ内は実行クロック数です(+はウェイト)。つまり，最高速でも16ドット描くのに26クロック+VRAMウェイト数だけかかります。CRTCのほうは16ドット描くのに8クロックし

かかりませんので，速度的には非常に不利です。

前のラスタを描き終わったかどうかのチェックは，水平ブランク信号が$E008のビット7に出ているので，Aレジスタに$7F，DEレジスタに$E008を入れておき，

```
        EX    DE,HL        (4)
LOOP:   CP    (HL)         (7+1)
        JP    NC,LOOP      (10)
        EX    DE,HL        (4)
```

とするのがもっとも高速だと思われます。水平ブランクを検出してから実際の処理に移るまでのロスは，CP命令の直後に水平ブランク期間になったときが最悪値なので，32クロックとなります。

これらをもとに，1ラスタあたりどれくらいが書き換え可能か計算します。CRTCに追いつかれないうちにできるだけ多く書き換えを行うため，ラスタの右側（つまり画面の右端）を書き換えます。で，CRTCが右端に行き着く前にどれだけ転送可能かというと，

$$(228-32)\div(26+2)=7$$

で，7×16＝112ドットです。実際にはCRTCは8ドットずつCGROMから取ってきて表示していることなども考えれば，6×16＝96ドットくらいが限界でしょう。

つまり，画面の右端に96×200ドットのグラフィックもどき表示ができることになります。

そうは問屋が卸さない

ところが実際にプログラムを書いて走らせてみると，グラフィックどころかラスタごとにキャラクタを書き換えている様子すら見られません。かろう

リスト1

```
/*
  MZ-700のキャラグラでグラフィックするための
  コンバートテーブルを作る
*/

#include <stdio.h>
#include <stdlib.h>

int printConvTbl(char *romst);
int findsamebit(char *romst, unsigned char bitpt);
int findsamebit2(char *romst, unsigned char bitpt);
int findnearbit(char *romst, unsigned char bitpt);
void readygetdistance(unsigned char b);
int getdistance(unsigned char b1, unsigned char b2);

static unsigned char CGROM[4096];
static int sd[8];

void main(int argc, char *argv[])
{
FILE   *fp;
int    i;
int cnt=0;

  fp=fopen("CGROM700.BIN","rb");
  if ( fp==NULL ) { printf ("CGROMファイルが読めません\n\n"); exit(-1); }
  fread(CGROM,1,4096,fp);
  fclose(fp);

  for ( i=0; i<8; i++ ) {
     cnt+=printConvTbl (CGROM+i);
     printf("\n");
  }
/* printf("Result: %d/2048\n",cnt);*/

  exit(0);
}

int printConvTbl(char *romst) {
  int i;
  int c;
  int cnt=0;
  int yoko=0;
  printf("{\n");
  for ( i=0; i<256; i++ ) {
     if ( yoko==0 ) printf("\t");
     c=findnearbit(romst,i);
     if ( c==-1 ) {
        printf("0xff,");
     } else {
        printf("0x%.2x,",c);
        cnt++;
     }
     if ( yoko==7 ) printf ("\n");
     yoko=(yoko+1)&7;
  }
  printf("};\n");
  return cnt;
}

int findsamebit(char *romst, unsigned char bitpt) {          /* キャラのみ */
  int i;
  unsigned int pt;
  for ( i=0; i<256; i++ ) {
     pt=(*romst) & 0x0ff;
     if ( pt==bitpt ) return i;
     romst+=8;
  }
  return -1;
}

int findsamebit2(char *romst, unsigned char bitpt) {         /* アトリビュート付き
*/
  int i;
  unsigned int pt;
  for ( i=0; i<256; i++ ) {
     pt=(*romst) & 0x0ff;
     if ( pt==bitpt ) return i;
     if (  ( pt ^ 0x0ff ) == bitpt ) return i;
     pt=(* (romst+2048)) & 0x0ff;
     if ( pt==bitpt ) return i;
     if ( ( pt ^ 0x0ff ) == bitpt ) return i;
     romst+=8;
  }
  return -1;
}

int findnearbit(char *romst, unsigned char bitpt) {
  int i,d;
  int min,mi;
  unsigned int pt;
  min=0x7fffffff; mi=0;
  readygetdistance(bitpt);
  for ( i=0; i<256; i++ ) {
     pt=(*romst) & 0x0ff;
     d=getdistance(pt,bitpt);
     if ( d==0 ) return i;
     if ( d<min ) { min=d; mi=i; }
     romst+=8;
  }
  return mi;
}

void readygetdistance(unsigned char b) {
  int i,j;
  unsigned char mask;
  unsigned char m0,mm;

  mask=0x01;
  for ( i=0; i<8; i++ ) {
     sd[i]=( (mask&b)!=0 ) ? 8 : 0;
     mask<<=1;
  }
  m0=b; mm=b;
  for ( i=0; i<8; i++ ) {
     m0=((m0<<1)|(m0>>1))& (mm^0x0ff);
     mm=(mm<<1)|(mm>>1)|mm;
     mask=0x01;
     for ( j=0; j<8; j++ ) {
        if ( (mask&m0)!=0 ) sd[j]=i+1;
        mask<<=1;
     }
     if ( mm==0x0ff ) break;
  }
}

int getdistance(unsigned char b1, unsigned char b2) {
  int i;
  int d;
  unsigned char mask=0x01;

  d=0;
  for ( i=0; i<8; i++ ) {
     if ( ( b1 & mask )!=( b2 & mask ) ) d+=((1<<8)+sd[i]);
     mask <<= 1;
  }
  return d;
}
```

じて画面のいちばん上，1ラスタ目だけに書き換えとおぼしきちらつきが見られるだけでした。表示データをいろいろ変えてテストしたところ，どうもCRTCに対して書き換えが全然間にあっていないようです。横表示ドット数を減らしていってもちっとも改善しません。

実は「VRAMウェイトが1クロック」というのは資料がなかったので勝手に仮定した値でした。書き換えが間にあってないということは，VRAMウェイトがもっとかかっているということです。

本来ならVRAMのウェイトを計測するプログラムを書いて計算をやり直すべきなんでしょうが，私はもうイヤんなってしまったので，水平ブランクも見ずに全力疾走で転送してCRTCに追い抜かれない横ドット数を探し，あとのタイミングはテストパターンを見ながらテキトーに命令を並べて調整しよう，と投げやりな気分でちまちまとテストしてしまいました。

そしたら横48ドットにしたところで調整もせずにあっさり映ってしまったのでした。

MZGRAPH

というわけでリスト2がMZ-700で48×200ドットのモノクログラフィック表示もどきを行うプログラムです。$C000からの1200バイトを画像とみなして表示します。処理としては，垂直表示期間の開始を検出したあとはひたすらPOP-LD転送でラスタごとのデータをガシガシと書き換えているだけです。前述のように水平ブランクも見ていないのですが，ちゃんと動いているみたいなのでよしとしています。ちなみにVRAMウェイトを逆算すると平均25クロック(表示期間だから？)となります。ほんとか。

サンプル画像に，電子手帳の手書きメモの落描きから適当に取ってきたものと，MZ-700のマニュアルのイラストをスキャナで取り込んだものを示します。もとがキャラクタだと思えば結構見られる絵になっているんじゃないでしょうか。

せっかく毎フレーム全CPUパワーを投入して描いているんだから，と簡単なアニメーションをやってみたのがリスト3です。といっても私はアニメなんてパラパラマンガしか経験がないので，5コマのいいかげんな絵がループしているだけですが。しかし適当アニメとはいえ，これがMZ-700で動いているのを見ていると意味もなく感動しますね。こちらは$B000から1200バイトずつが各フレームの画像データになっています。

なお，なにかキーを押すと終了，のような気のきいた作りにはなっていませんので，飽きたらリセットしてください。

おしまい

はるか昔，Oh!MZ誌上でPCG-700を利用した128×128ドットのグラフィック表示がありましたが，やっぱり無改造のMZ-700でやるのはまた違った感動があります。X1やPC-8801が羨ましくてしかたなかったガキンチョ時代の想いもこれでちょっと解消。1024×768画素フルカラーが当たり前の人からすれば「あなたはなにをいっているのですか」と思われるでしょうが，そんなことをいう奴はキャラグラの野球拳(byハドソン)5万回の刑に処してやる。ではさようなら。

リスト2

```
;
;  MZ-700でグラフィックもどき
;

HBLNK  EQU 0E008H
VBLNK  EQU 0E002H
IMAGE  EQU 0C000H

   ORG 1200H

   DI
   JR  ENTRY

LPEND0:
   LD  HL,VBLNK             ;表示プログラムの末尾
   LD  A,7FH
L999:  CP  (HL)
   JR  C,L999
   JP  LOOP
LPEND1:

ENTRY:
   LD  HL,0D822H            ;アトリビュート画面初期化
   LD  DE,34
   LD  A,07H                ;白地に黒
   LD  B,25
L11:   LD  (HL),A
   INC HL
   LD  (HL),A
   INC HL
   LD  (HL),A
   INC HL
   LD  (HL),A
   INC HL
   LD  (HL),A
   INC HL
   ADD HL,DE
   DJNZ    L11

   LD  HL,START             ; 表示プログラムを生成する
   LD  DE,0D000H+34
   LD  A,25
   LD  C,7              ; 1行目の1ラスタ目は生成済み
   JR  LP1
LP2:   LD  C,8
LP1:   LD  B,3
   PUSH    DE
LP0:   LD  (HL),0E1H          ; POP   HL
   INC HL
   LD  (HL),022H            ; LD    (nn),HL
   INC HL
   LD  (HL),E
   INC HL
   LD  (HL),D
   INC HL
   INC DE
   INC DE
   DJNZ    LP0
   POP DE
   DEC C
   JR  NZ,LP1
   LD  BC,40
   EX  DE,HL
   ADD HL,BC
   EX  DE,HL
   DEC A
   JR  NZ,LP2

   EX  DE,HL                ; プログラムの終端をコピー
   LD  HL,LPEND0
   LD  BC,LPEND1-LPEND0
   LDIR

   LD  HL,VBLNK             ;垂直ブランク期間の開始を待つ
   LD  A,7FH
L1:    CP  (HL)
   JP  NC,L1
L2:    CP  (HL)
   JP  C,L2

LOOP:  LD  HL,IMAGE
   LD  SP,HL                ;イメージデータ開始アドレス

   POP HL                   ;1ラスタ目を描いておく
   LD  (0D000H+34),HL
   POP HL
   LD  (0D000H+36),HL
   POP HL
   LD  (0D000H+38),HL

   LD  HL,VBLNK             ;スタート待ち
   LD  A,7FH
L3:    CP  (HL)
   JP  NC,L3                ;垂直ブランク期間の終わりを待つ

   LD  IX,(ENTRY)           ;ウェイト

START:                      ;この後は自動生成される

   END
```

MZ-700エミュレータ

Column

今回CD-ROMに収録されたプログラムではMZ-700用のメモリイメージファイル形式としてまるくん氏作のMZ-700エミュレータmz700winで使用されているファイルフォーマットが採用されています。ついでというわけではありませんが，mz700winもCD-ROMに収録されています。

当然のことながら，収録されているのはエミュレータ本体のみで，システムプログラムやBIOS ROMイメージは付属しておりません。実機をお持ちの方のみご使用ください。ただし，ROMの吸い上げ方などは聞かないように。

とはいうものの，MZ-700のBIOS ROMはわずか4KBのものにすぎません。おまけにマニュアルにソースがついているという親切ぶり。まっとうなZ80プログラマなら，初期マイコン時代に作られたBIOSよりもずっと効率のいいプログラムにできるはずですし，大半のプログラムの動作にはBASICなどのシステムプログラムはほとんど必要なかったというのも強みです。

ということで，オリジナルコンパチBIOSの投稿あるいは開発ボランティアを募集します。秋号では，ちゃんと動くかたちでエミュレータが掲載できるといいのですが。

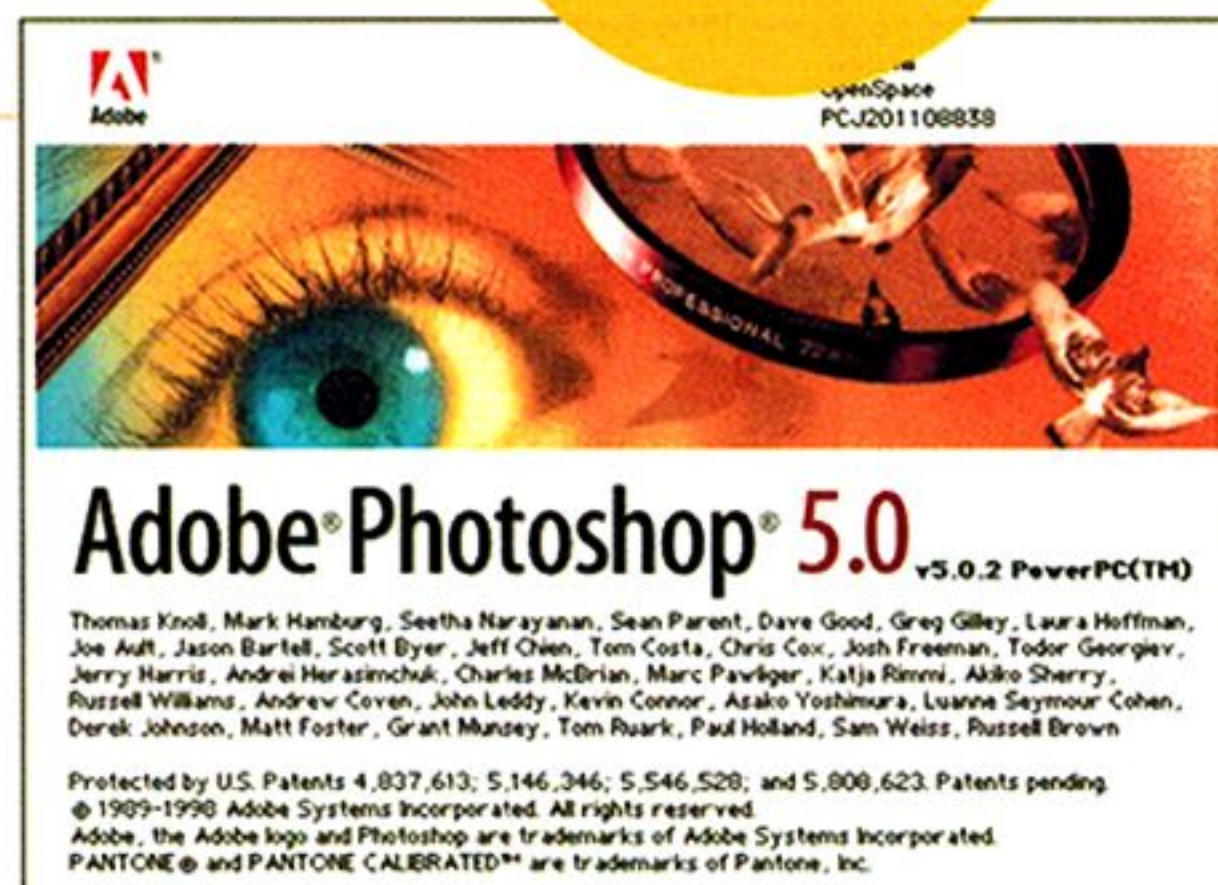
リスト3

```
;
;  MZ-700でグラフィックもどき(アニメもどき)
;

HBLNK  EQU 0E008H
VBLNK  EQU 0E002H
IMAGE  EQU 0B000H

   ORG 1200H

   DI
   JR  ENTRY

LPEND0:
   LD  HL,VBLNK              ;表示プログラムの末尾
   LD  A,7FH
L999:  CP  (HL)
   JR  C,L999
   DEC B
   JP  NZ,LOOP
   INC DE                    ;次のフレーム
   INC DE
   LD  A,(DE)
   OR  A                     ;最初に戻る
   JR  NZ,L998
   LD  DE,FRAMETBL
   LD  A,(DE)
L998:  LD  B,A               ;繰り返し回数
   INC DE
   JP  LOOP
LPEND1:

FRAMETBL:                    ;アニメーションのタイミングデータ
   DEFB    12
   DEFW    0B000H
   DEFB    6
   DEFW    0B4B0H
   DEFB    6
   DEFW    0B960H
   DEFB    6
   DEFW    0BE10H
   DEFB    12
   DEFW    0C2C0H
   DEFB    6
   DEFW    0BE10H
   DEFB    6
   DEFW    0B960H
   DEFB    6
   DEFW    0B4B0H
   DEFB    0

ENTRY:
   LD  HL,0D822H             ;アトリビュート画面初期化
   LD  DE,34
   LD  A,07H                 ;白地に黒
   LD  B,25
L11:   LD  (HL),A
   INC HL
   LD  (HL),A
   INC HL
   LD  (HL),A
   INC HL
   LD  (HL),A
   INC HL
   LD  (HL),A
   INC HL
   LD  (HL),A
   INC HL
   ADD HL,DE
   DJNZ    L11

   LD  HL,START              ; 表示プログラムを生成する
   LD  DE,0D000H+34
   LD  A,25
   LD  C,7                   ; 1行目の1ラスタ目は生成済み
   JR  LP1
LP2:   LD  C,8
LP1:   LD  B,3
   PUSH    DE
LP0:   LD  (HL),0E1H             ; POP   HL
   INC HL
   LD  (HL),022H             ; LD    (nn),HL
   INC HL
   LD  (HL),E
   INC HL
   LD  (HL),D
   INC HL
   INC DE
   INC DE
   DJNZ    LP0
   POP DE
   DEC C
   JR  NZ,LP1
   LD  BC,40
   EX  DE,HL
   ADD HL,BC
   EX  DE,HL
   DEC A
   JR  NZ,LP2

   EX  DE,HL                 ; プログラムの終端をコピー
   LD  HL,LPEND0
   LD  BC,LPEND1-LPEND0
   LDIR

   LD  HL,FRAMETBL
   LD  B,(HL)                ; 繰り返し回数
   INC HL
   EX  DE,HL                 ;(DE):イメージデータ開始アドレス

   LD  HL,VBLNK              ;垂直ブランク期間の開始を待つ
   LD  A,7FH
L1:    CP  (HL)
   JP  NC,L1
L2:    CP  (HL)
   JP  C,L2

LOOP:  LD  A,(DE)                ;イメージデータのアドレスを得る
   LD  L,A
   INC DE
   LD  A,(DE)
   LD  H,A
   DEC DE
   LD  SP,HL                 ;SP:イメージデータ開始アドレス

   POP HL                    ;1ラスタ目を描いておく
   LD  (0D000H+34),HL
   POP HL
   LD  (0D000H+36),HL
   POP HL
   LD  (0D000H+38),HL

   LD  HL,VBLNK              ;スタート待ち
   LD  A,7FH
L3:    CP  (HL)
   JP  NC,L3                 ;垂直ブランク期間の終わりを待つ

   LD  IX,(ENTRY)            ;ウェイト

START:                       ;この後は自動生成される

   END
```

Adobe Photoshopでゲームを作ろう

番外編

古籏一浩 Furuhata Kazuhiro

グラフィックツールとして完全に定番となった感じのあるPhotoshop。非常に多機能なツールとして知られているが，そこに搭載された多彩な機能を組み合わせればPhotoshopだけでゲームを動かすこともできる。ここではレイヤーを使ったパズルゲームについて解説してみよう。

Photoshopでゲームが作れるか？

「Adobe Photoshopでゲームを作ろう」とありますが，Photoshopでゲーム用の画像を作ろう……というものではありません。Photoshop上でゲームを動かそうという試みです。

とはいうものの，「はてさてPhotoshopでゲームが作れるものかはなはだ疑問だ」という方は多いと思います。Photoshopを使って作られたゲームは数あれどPhotoshopそのものでゲームをする，というのは前代未聞です。Photoshopを買った人で「Photoshopでゲームを作ろう」と考える人はいないでしょう。Photoshopを知らない人のために簡単に説明すると，Photoshopはフォトレタッチ処理を行うソフト，画像処理を行うソフトです。画像処理だけでなくCGを描くといった用途にも使用されています。もちろんプログラム言語はおろかスクリプトも搭載されていません。

しかしPhotoshopは「レイヤー」機能とレイヤー同士の「演算機能」（図1）を備えています。この2つをうまく使えばPhotoshopでゲームを作ることが可能になります。そうパズルゲームであればPhotoshopで動かすことも可能なのです。

レイヤー機能と演算機能

ゲームを作る前にレイヤーと演算機能について説明します。レイヤーというのは透明なガラス（アニメーションを知っている人ならセルといったほうがわかりやすいでしょう）です。この上に色をつけたりフィルタを使って特殊効果を施すことになります。

演算機能は多少複雑です。Photoshop（ver.5）は以下の17の演算機能を備えています。

[1] 通常
[2] ディザ合成
[3] 乗算
[4] スクリーン
[5] オーバーレイ
[6] ソフトライト
[7] ハードライト
[8] 覆い焼きカラー
[9] 焼き込みカラー
[10] 比較（明）
[11] 比較（暗）
[12] 差の絶対値
[13] 除外
[14] 色相
[15] 彩度
[16] カラー
[17] 輝度

この演算機能（演算モード）ですが，Photoshop ver.5のマニュアルにはほとんど記載されていません。Photoshop ver.2.5のマニュアルには演算式付きで，Photoshop ver.3.0のマニュアルには言葉で一応演算式が説明されています。今回はゲームに使用する演算モードについて説明します。

[1] 通常

現在のレイヤーのピクセルをそのまま表示し，下のピクセルは無視され反映されません。

R＝dR
G＝dG
B＝dB

[3] 乗算

2つのレイヤーのピクセル値を乗算し255で割った値がピクセル値になります。灰色と黒であれば（128×0）÷255＝0のようになります。論理演算でいえば論理積（AND）になります。以後の説明では乗算には&記号を使っています。

R＝（dR＋sR）÷255
G＝（dG＋sG）÷255
B＝（dB＋sB）÷255

[4] スクリーン

2つのレイヤーのピクセル値を反転し乗算する，とマニュアルに書いてありますが，実際にやってみると論理和（OR）そのものです。

R＝dR | sR
G＝dG | sG
B＝dB | sB

[12] 差の絶対値

2つのレイヤーのピクセル値の差分を取り正数化します。

R＝absolute（dR－sR）
G＝absolute（dG－sG）
B＝absolute（dB－sB）

以上です。ほかの演算モードについてはマニュアルやPhotoshopの参考書籍を見てください。ただ，マニュアルおよび参考書籍では演算モードの説

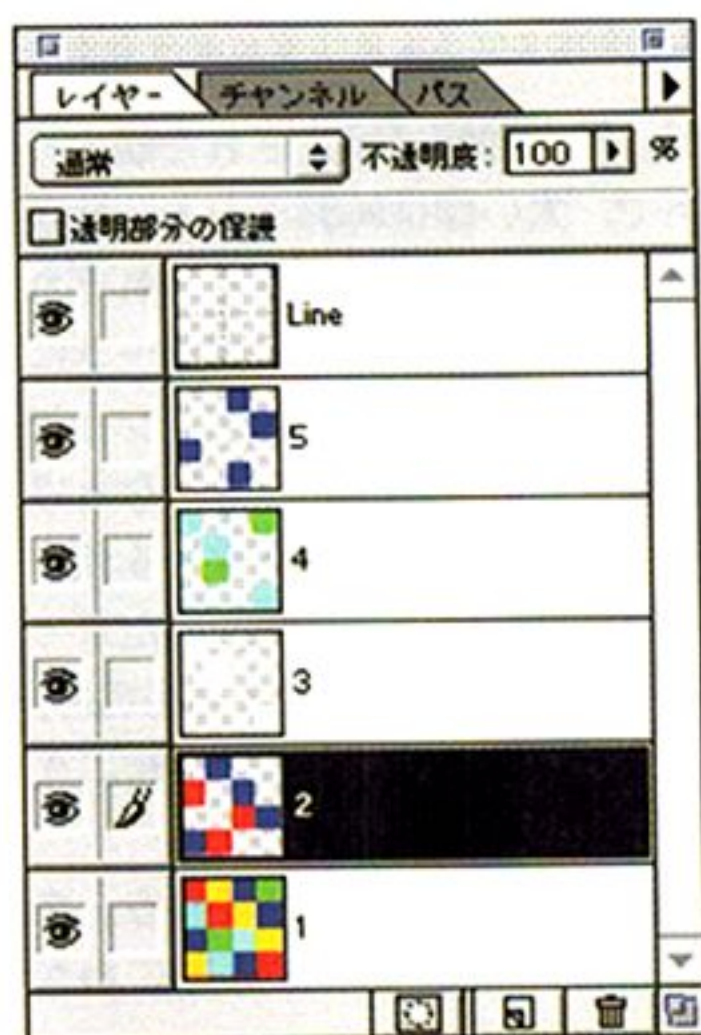
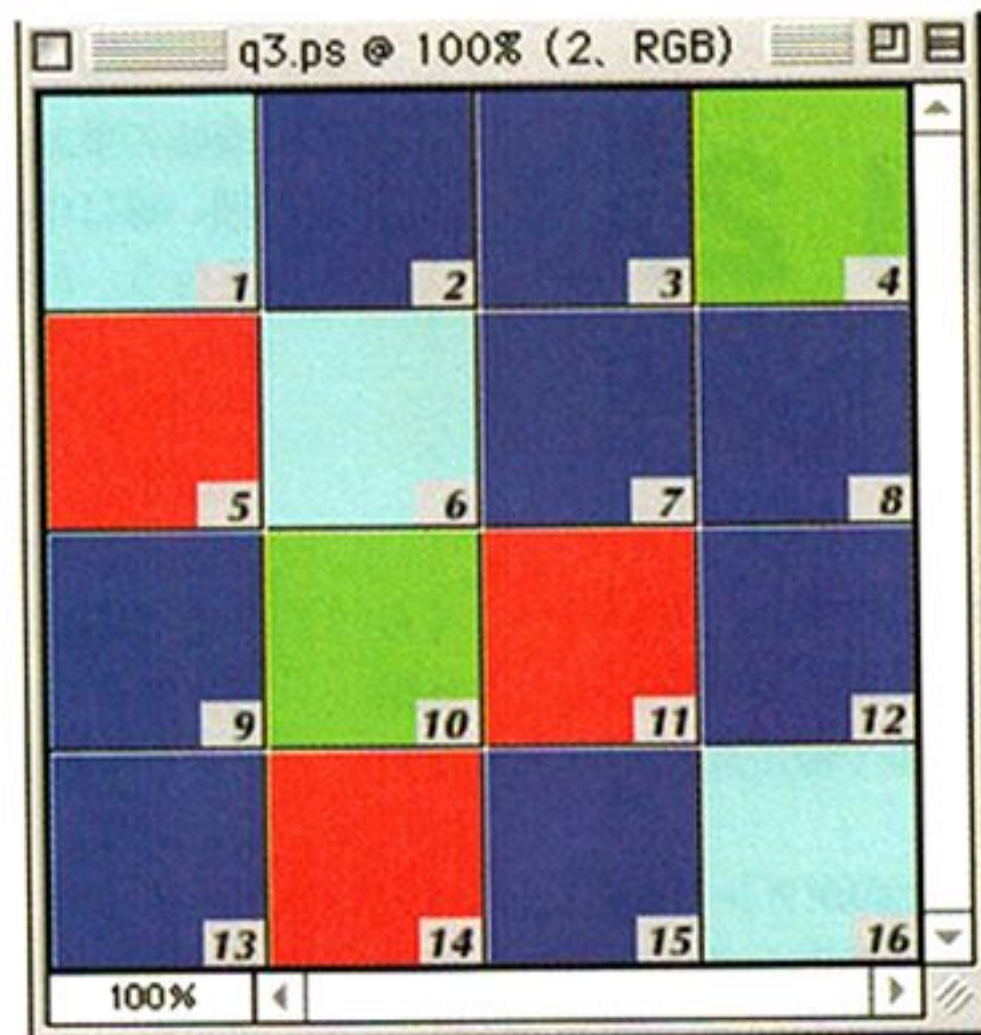



図1

図2

明が少し変です。説明のとおりに解釈して計算すると実際の結果とは異なるピクセル値になってしまいます。あと，焼き込みツール/覆い焼きツールは演算誤差のせいか白と黒しかないピクセルでも色がついてしまいます。

ほかにも彩度とかゲームに使えるモードがありますが，とりあえず簡単なものを作ってみましょう。

Black 16

ゲームとしては16のマス目の色を全部黒色にしたらクリアとしましょう。全部黒くしても「見事にクリアしました！」なんてメッセージは出ません（それは不可能)。それでは以下に作り方を説明します。CD-ROMには一応マスターファイルを入れてあります。図2は作成したパズルのゲーム画面です。

まずマス目を作り16に分けましょう。あとは，分かれた16のマス目にペイント缶で色を塗ります。いくつかレイヤーを作って演算モードを組み合わせて16のマス目を全部黒になるようにします。演算モードを理解していないとちょっと難しいでしょう。演算は下のレイヤーから行われます。たとえば，以下の3つのレイヤーがあり，カッコ内の色だとしましょう。

レイヤー3（水色）
レイヤー2（白色）
レイヤー1（赤色）

演算はまずレイヤー1とレイヤー2で行われます。その演算結果とレイヤー3で演算が行われます。この順序と法則を忘れては駄目です。この法則と演算モードを組み合わせてやれば，ちょっとしたパズルとなるわけです。

上記の3つのレイヤーに演算モードを設定して黒にするには以下のようになります。

レイヤー3（水色）［差の絶対値］
レイヤー2（白色）［差の絶対値］
レイヤー1（赤色）［通常］

これで黒になります。また，以下のような組み合わせでも黒になります。

レイヤー2（水色）［乗算］
レイヤー1（赤色）［通常］

これだとわかりにくいかもしれません。実は各色に番号をつけるとわかりやすくなります。

色名：番号：ビット列
黒色：　0：000
青色：　1：001
赤色：　2：010
紫色：　3：011
緑色：　4：100
水色：　5：101
黄色：　6：110
白色：　7：111

最初の3つのレイヤーの場合は差の絶対値ですから，

赤色－白色－水色＝黒色
2－7　　＝－5
－5の絶対値＝5
5－5　　＝0

となり黒となります。2つのレイヤーの例では，

赤色＆水色＝黒色
2＆5　＝0

となります。

「なんで，そうなるの？」

という人は2と5をビット列に直すとわかりやすいかもしれません。

2：010
5：101
―――
0：000

ここで両方とも1でない場合は結果は0（黒色）になります。つまりどちらも同列が1になっているところがないので0になります。これが青色と水色なら青色になります。

1：001
5：101
―――
1：001

スクリーンならば両方とも0の場合のみ1となります。赤色（2）と水色（5）では以下のようになります。

2：010
5：101
―――
7：111

つまり白色になります。青色と水色のスクリーン演算であれば青色になります。わからない人は1と0に直して上記規則に従って紙に書いてみるとよいでしょう。

このような組み合わせをレイヤーとしていくつか作ればパズルゲームのできあがりです。

＊

CD-ROMにサンプルとして3つ用意しておきました。演算モードを使ったパズルはほとんど存在しないので，解答ファイルも入れてあります。

ゲームを作るにはC/C++言語を覚えたり，開発するためのソフトを別途用意しなければいけないということは決してありません。極端な話，Photoshopのようなグラフィックツールでさえゲームを作ることができました。HTMLとスクリプトは当然として，マクロ言語のある表計算ソフトやワープロソフトなど，手元にあるソフトでもアイデア次第でゲームを作ることは可能なのです。

Vmaker Professional 2.0

アイフォー(CD-ROMに体験版を収録)

Vmaker Professional 2.0(以下Vmaker)はベクタライズソフトである。ベクタライズというのはビットマップの画像をベクトルデータの画像にコンバートする作業を指している。ビットマップデータをベクトルデータに変換することによって，FLASHなどを使ったWebでの利用が容易になったり，CADソフトでのエディットが可能になるというわけだ。なによりも「点の集合体」を「点と線の集合体」に変換するので，データサイズが小さくなる，拡大/縮小といった編集作業でもデータが劣化しないなどのメリットを享受できるのだ。

●下絵の取り込みと修正

まずはベクタライズするための下絵を読み込む。読み込みはTWAIN機器からの入力(スキャナなど)またはファイルからの入力(サポートフォーマットはBMP, PCX, TIFF, UBB)のどちらかを選択して行う。Vmakerでは基本的に白黒2値のデータしか扱わないので、カラー原稿などの場合は読み込んだあとカットオフ設定で白黒2値に変換する必要がある。また例外的に16色のデータも扱えるようになっているが、その場合はあらかじめ16色で保存したBMPなどのファイルが必要となる。16色原稿の場合、ベクタライズ後は16色それぞれの色情報を持ったベクトルデータが生成されることになる。

読み込んだ下絵データは線がかすれていたり、途切れていたりするので修正を施さないと綺麗なベクトルデータは得られない(別にベクタライズしてから修正を加えてもよいのだが、やはり下絵の段階で綺麗にしておけばそれなりによいデータが得られるので手抜きは禁物)。ということで、読み込んだ下絵に対して修正を加えるため、Vmakerには必要最低限の描画機能が備わっている。用意されているのは点，直線，矩形，楕円，塗りつぶしの各コマンド。ブラシサイズは1～16ドットの間で任意に選べる。これ以上ないというくらいシンプルなメニューだが，別にVmakerで一から絵を描くわけではなく，あくまで修正用に用意されたものだからこれで十分だろう。

ただひとつだけ不満だったのがアンドゥ機能の欠如。アンドゥがないために修正に失敗したりすると，また最後にセーブした時点まで戻らないといけない仕様になっている。これは緊張感を持って仕事をしろ，ということなのだろうか？ そういう労力・束縛を排除することがこういったソフトの役割だと思うのだが……。

またVmakerは描画機能のほかにフィルタ機能も備えている。これは下絵にあるゴミを消去したり，途切れた線をつないだりといった作業を一括して行うもので，よりベクトルデータの生成のしやすいビットマップイメージを作る，というのが目的の機能だ。だが実際に使ってみると，自分が残そうと思っていた線が消されてしまったり，逆にいらない線が補強されてしまったりとなかなか思いどおりの結果を得るのは難しい。フィルタ機能は実行時に指定できるパラメータなどもないので，フィルタ自体のクセを覚えて使っていくしかないだろう。このほかにも左右・上下反転，90度回転機能などの基本的なエディット機能が装備されている。

●3モードのベクタライズモード

さていよいよ肝心のベクタライズ(なぜベクタライズのアイコンが茄子なのだろうか？ 謎だ)。Vmakerは中心線モード，輪郭モード，シンクロモードといった3モードのベクタライズモードを持っている。中心線モードは図形の線の中心を，輪郭モードは図形の輪郭を抽出してベクタライズを行う。設計図や天気図など主に線で構成された画像をベクタライズするときは中心線モードを，イラストやロゴなど広く塗りつぶされた領域が多い画像は輪郭モードを使うのが一般的だろう。シンクロモードは中心線・輪郭モードの両方の機能の兼ね備えており，最大線幅で指定された値より細い線は中心線モードで，太い線は輪郭モードで処理を行う。市松模様をベクタライズしてみるとそれぞれのモードの効用がひと目でわかる。また輪郭モードにおいてはドットトレースを行うかどうかのオプションも設定できる。ドットトレースを行うと，通常ならばゴミとして処理される1ドットだけのデータも4点からなる矩形のベクトルデータに変換される。したがってベクタライズ後のデータは飛躍的に増大するものの，1ドットももらさずベクタライズすることが可能になるわけだ。

ベクタライズの際にはさまざまな設定ができる。まずベクトルレベル。粗いから細かいまで5段階にわたる設定が可能だ。

もっと細かい設定をしたい場合は詳細設定で鮮鋭化，曲線化レベル，ノイズ幅，歪み幅，ベクタライズ容量をそれぞれ指定することができる。鮮鋭化は角および交点の歪みをどれくらい補正するか，曲線化レベルはどこまでを曲線と認識するか，ノイズ幅はどの大きさのノイズまでをゴミとして除去するか，歪み幅は線の歪みをどの程度補正するかを指定することができる。扱う画像によって試行錯誤しながらパラメータを設定していくことが必要だろう。できれば設定したパラメータ群に名前をつけて保存できるような機能がほしかったところだが，残念ながらそのような機能はついていない。

ベクタライズされた画像は元の画像と重ね合わせて表示されるので，元画像とベクトルデータの比較が簡単にできる(重ね合わせ濃度は通常属性レイヤーで70%といったところ？)。ベクタライズされたデータはもちろんベクトルデータ用ツールボックスで修正を施すことができる。修正はハンドルマークをドラッグするだけ，というシンプルなものなので特にとまどうようなこともない。修正以外にも新たにハンドルマークを追加して，曲線や直線，矩形・楕円などを描画することができる。もちろんこのモードでもアンドゥなんて甘っちょろい機能はなしだ。修正を終えたデータはDXF,EPS,WMF,C6,V5いずれかのフォーマットで出力することができる(C6,V5はi4社製CADソフトCANDY用フォーマット)。

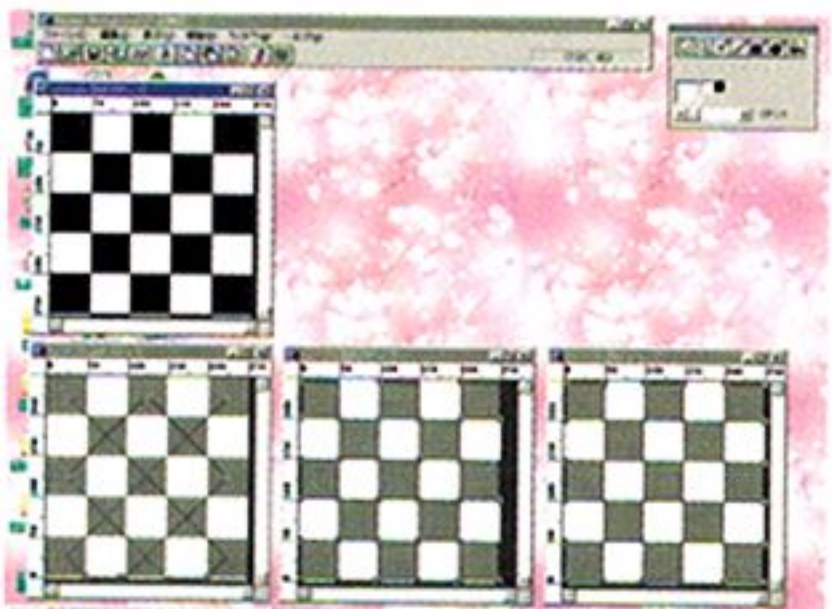
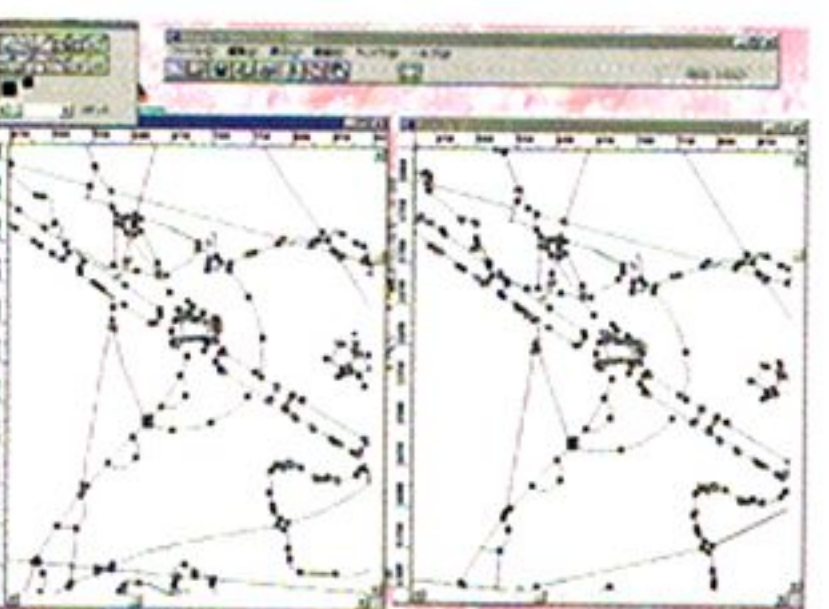
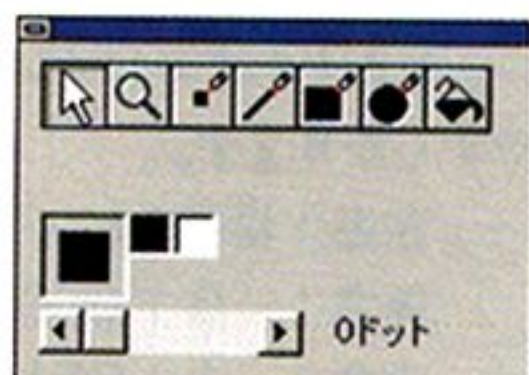
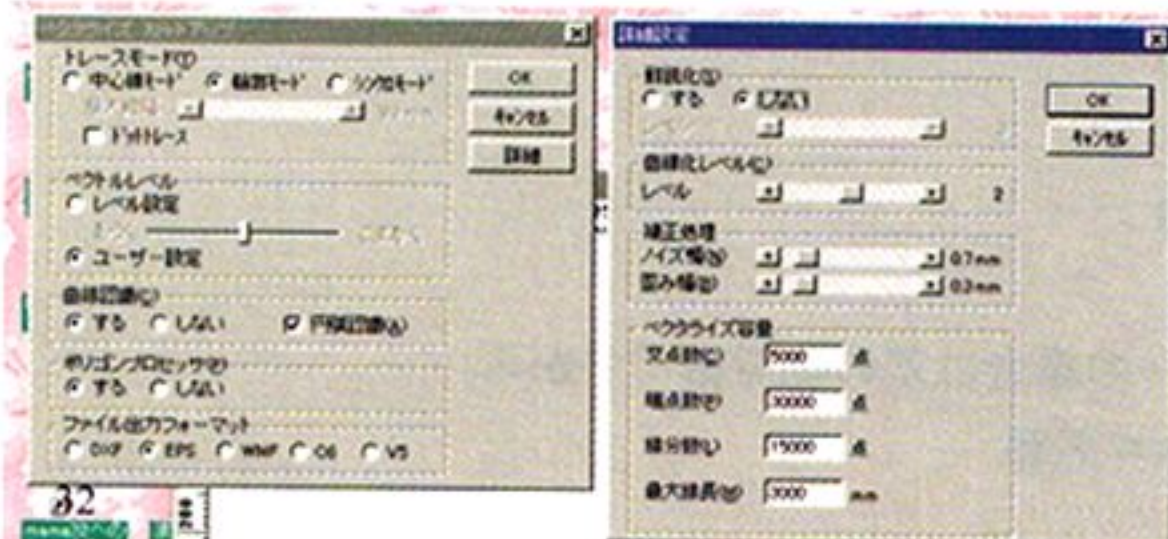
(高橋哲史)

カットオフ設定の様子。どこまでを黒として認識するかを指定する。プレビューはリアルタイムに変化する

カットオフ終了後の画面

16色のイラスト画像を輪郭モードでベクタライズしてみたところ。右上はドットトレースOFF，右下はON。……やはりどちらもかなり修正を施さないと使えそうにない感じ。トレーススピードは結構速く，両画像とも2秒ほどでベクタライズを完了した(K6/200MHz,64Mバイトメモリのマシンを使用)

イメージデータ用ツールボックス

イメージデータ用ツールボックス

それぞれ異なるモードでベクタライズしたところ。下段，左から中心線モード，輪郭モード，シンクロモード

天気図をそれぞれベクトルレベルを変えてベクタライズしてみた。左が「細かい」，右が「粗い」で処理してある。ハンドルマークの多さ，曲線の滑らかさに違いを見てとることができる

ベクタライズの詳細設定画面

Step to the Black Arts

プログラムを作ることが難しくなっている状況のひとつに，「ちょっとした用途」のツールが作りにくくなっていることがあるかもしれない。
データ構造はしっかり公開されているのだけれど，仕様を見るだけでうんざりするとかいうこともある。タグやチャンク構造のデータは慣れないとちょこちょこっといじろうにも手が出しにくく見える。ちゃんと考えられた「構造」なのだが，構造を理解するところから始める必要がある。
用途が高度化しているのも問題だ。DirectXなどはGameSDKという前身が示すように，ゲームを作るために用意されているライブラリ群だが，ゲームを作ろうと肩肘張っていると道は遠い。SDKを扱うにもまず資料が必要になる。いろいろとややこしいのだ。サンプルをいじって遊ぶところから始めないとなかなか先に進めないだろう。できることの幅は広がっているのだが，第一歩が少し大変になっているのだ。
手軽な題材はないだろうか？　インターネット時代に，CGIなどはよい題材なのだが，サーバを確保しなければいけないのは大きな障害となる。
JavaScriptなどはもっとも手軽で応用範囲も広いのだが，互換性の問題があり，本質以外のところでいろいろ面倒だ。テキストエディタとWebブラウザだけあればできるとはいうものの，実際に開発環境として使ってみるとはなはだ使いにくいことがわかる。せめて適切なエラーを出してくれればよいのだが……。
周辺環境も整備していかなくてはならない。グラフィックパターンを作るのに適切なツール，テクスチャを描くのに適したツール，そう，3Dということになれば，モデリングツールが必要になる。それも「ゲームに適したもの」が必要だ。たいていの場合は「大は小を兼ねる」ものだが，そうなると初期投資は馬鹿にならない。
結局，地道にツールから揃えていくというのがいちばんなのかもしれない。

Future BASICによるMacintoshプログラム制作　第3回
ファイル処理に挑戦しよう

古籏一浩　FuRuhata KaZuhiro

今回はファイル処理について説明する。リソースが絡まなければMacintoshも普通のファイル操作を行うだけで簡単にデータの読み込み/保存ができる。扱う命令も比較的簡単なものばかりで，応用範囲がきわめて広いのでぜひマスターしておきたい。

テキストファイルを読み込む

最初は簡単なファイル処理から行ってみましょう。読み込むためのファイルは「テキストファイル」(文字だけのファイル)とし，ファイル名も決めておきます。この場合の手順は以下のようになります。

[1] ファイルを開く(ファイルオープン)
[2] データを読み込む
[3] ファイルの終わりになるまで[2]へ
[4] ファイルを閉じる(ファイルクローズ)

それでは命令に対応させていきましょう。まず[1]のファイルを開くにはOPEN命令を使います。OPEN命令の書式は以下のようになります。

```
OPEN "入出力形式", ファイル番号, ファイル名, レコード長, ボリューム参照番号
(例：OPEN "I",#1,"sample.txt",,volRefNum%)
```

入出力形式はファイルをどのように扱うか，を指定するもので以下の文字を指定することができます。

I = 読み込み専用
O = 書き込み専用(ファイルが存在する場合は上書き)
A = 追加書き込み(ファイルの末尾にデータが追加される)
R = ランダムアクセス
N = ネットワークランダムアクセス(同時読み書き可能)

ファイル番号は便宜上つけるもので1～99(最大)までの数字を指定します。#はつけてもつけなくても構いません。本講座ではファイル番号に関しては#を付加して説明します。またファイル番号の最大値はプリファレンスダイアログで指定します(図1)。

ファイル名は開きたいファイル名です。最大31文字までです(HFSの場合。HFS＋だと255文字までだと思われるが未確認)。レコード長はランダムアクセス以外は指定を省略して構いません。

ボリューム参照番号はファイル位置を指定するものです。ボリューム参照番号は一時的に使用されるもので永続的なものではありません。再起動したり電源を落とせば変わってしまいますので注意してください。

ボリューム参照番号は省略することができます。省略時は0になります。逆に0を指定した場合どうなるかというと「現在のアプリケーションが存在するフォルダ」を基準としてファイルを参照するようになります。これについては次の「ファイルのパス指定」で説明します。

次に[2]です。データを読み込むにはINPUT命令を使います。ただし，この命令が使えるのは「テキストファイルである」という大前提が必要です。とりあえず最初に作成するものはテキストファイルとしていますので問題ありません。書式は以下のようになります。

```
INPUT #ファイル番号, 代入用文字列変数1, 代入用文字列変数2,……代入用文字列変数n
(例：INPUT #1,A$)
```

ただしINPUT文では「,(カンマ)」「EOF(ファイルの終わり)」「改行」があると，そこまでしかデータを読み込みません。また文字列の長さが255文字以上(途中改行がない)の場合は255文字までしか一度に読み込みません。1行単位で読み込みたい場合はLINE INPUT#を使います。この場合は以下の書式になります。

```
LINE INPUT #ファイル番号, 代入用文字列変数
(例：LINE INPUT #1,A$)
```

次に[3]のファイルの終わりになるまで[2]を繰り返す処理ですが，ファイルの終わりかどうかを調べる必要があります。ファイルが終わりかどうかを調べるにはEOF命令を使います。EOF命令の書式は以下のようになります。

```
EOF(ファイル番号)
(例：EOF(1))
```

もしファイルエンドであれば_true，そうでなければ_falseを返します。通常はEOF命令とWHILE～WEND命令を組み合わせて以下のような記述をします。

```
WHILE NOT EOF(ファイル番号)
 ファイル処理
WEND
```

NOT EOFはファイルエンドでなかったら繰り返すという意味で書かれ

プリファレンス
浮動小数点演算：
12 倍精度　◉ BCD
6 単精度　○ SANE
8 指数　○ 固定
49 四捨五入　○ その他
☑ 式を整数として最適化
変数：
☐ 配列要素を1から始める
初期設定の変数タイプ：整数
ファイル：
99 開ける最大ファイル数 (0-99)
ランタイム・エラー・チェック：
☐ 文字列/配列オーバーフローをチェック
エディタ：
☑ キーワードを大文字に変換
☑ 現在の行を囲んで表示
☑ キーワードにスペースが必要
☐ 行番号表示
2 プログラム構造のインデント
8 tab キーによる移動
48 フローティング・リマーク列
ﾌｫﾝﾄ：Osaka－等幅
ｻｲｽﾞ：12
出荷時設定に戻す　ｷｬﾝｾﾙ　OK

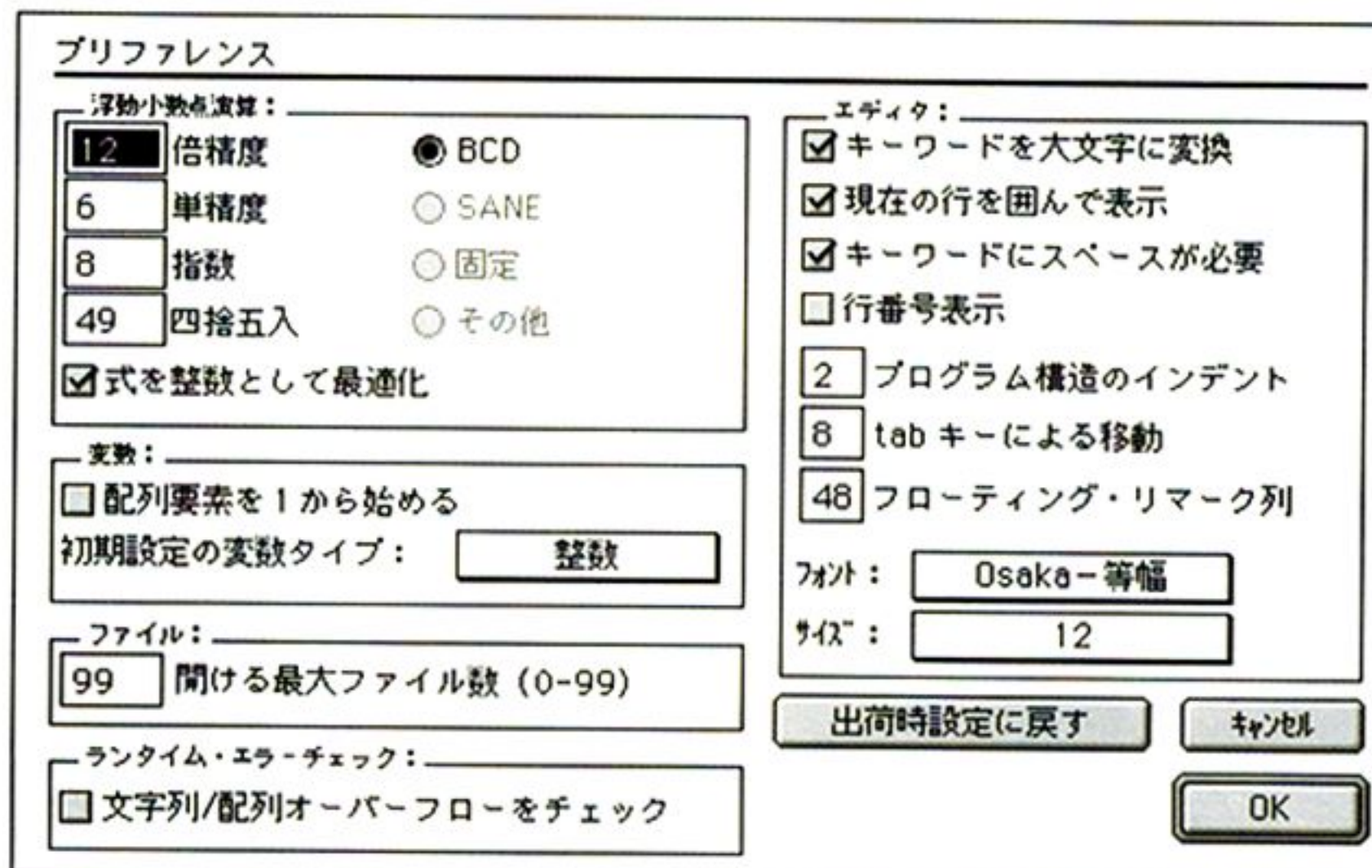



図1　最初は開くことができるファイル数は2になっているので多めに設定しておいたほうがよい

リスト1 sample1.bas

```
'-----------------------------------------------------
' "テキストデータを読み込んでウィンドウに表示する"
'-----------------------------------------------------
CALL GETFNUM("Osaka",fontID%)
TEXT fontID%
OPEN "I",#1,"sample.txt"
WHILE NOT EOF(1)
  INPUT #1,txt$
  PRINT txt$
WEND
CLOSE #1

BEEP
DO
  HANDLEEVENTS
UNTIL _false
```

ています。WHILE EOF(ファイル番号)= _falseでも間違いではありません。コンピュータ言語を扱っている本のなかには，このようなところはかなりさりげなく記述されていて，なぜそういう表記になっているかはあまり書かれません。

最後に[4]のファイルを閉じる処理ですが，これはCLOSE命令を使います。書式は以下のようになります。

CLOSE #ファイル番号

もし複数のファイルを閉じたい場合はCLOSE #1,#2のように「,」で区切って列記します。すべてのファイルを閉じたい場合はCLOSEと記述すれば自動的に開かれているファイルは閉じられます。またFuture BASICは終了時に開かれているファイルがあればCLOSEを記述しなくても自動的にファイルを閉じてくれます。

これで対応する命令がわかりましたので実際のプログラムを見てみましょう。sample1.basがテキストファイルを読み込んでウィンドウに表示するプログラムです。上記の説明と照らしあわせればどうなっているのかがわかると思います。

あと最初の2行について補足説明しておきます。Future BASICではアプリケーションを作成してプログラムを実行させたとき，Future BASIC上から実行した場合と異なり，日本語が化けてしまうことがあります。このため日本語システムには必ず組み込まれている大阪フォントを指定しています。大昔はフォントのIDは固定でしたが，フォントが予想以上に多くなりID管理するのが難しくなったため現在は「フォント名からIDを求めて使う」という具合になっています。つまりフォントIDは固定ではないということです。実際のところシステムで使われているフォントはIDが固定状態ですからTEXT 16384のように直接IDを指定しても大阪フォントにはなります。Macintoshの場合，OSの更新が頻繁に行われますから推奨されない方法は避けるべきです。そうしないとOSのバージョンアップでシステムエラーの憂き目にあってしまいます。

■ファイルのパス指定

sample1.basはアプリケーションと同じフォルダにあるテキストファイルを読み込んで表示しました。それでは，ひとつ下のフォルダにテキストファイルがある場合は，どうしたらよいのでしょう。ボリューム参照番号の値を設定すればよいのでしょうか？　フォルダ名と階層構造が決定している場合は：(コロン)を使ってファイルのパス(位置)を指定することが可能です。

Macintoshでは以下のようにファイルの位置指定を行うことができます。なおファイル名はtext.txt，下のフォルダはsub，ハードディスク名はMac HDDとします。

[1]同じフォルダ(カレントディレクトリ)
text.txt

リスト2 sample2.bas

```
'-----------------------------------------------------
' "テキストデータを読み込んでウィンドウに表示する"
'-----------------------------------------------------
CALL GETFNUM("Osaka",fontID%)
TEXT fontID%
OPEN "I",#1,"sample.txt":                    ' "同じ階層"
'OPEN "I",#1,":text:sample.txt":             ' "下の階層"
'OPEN "I",#1,"HD:text:sample.txt":           ' "絶対パス"
'OPEN "I",#1,"::text:sample.txt":            ' "上の階層"
WHILE NOT EOF(1)
  INPUT #1,txt$
  PRINT txt$
WEND
CLOSE #1

BEEP
DO
  HANDLEEVENTS
UNTIL _false
```

[2]ひとつ下のフォルダ(sub)にある場合(サブディレクトリ)
:sub:text.txt

[3]ひとつ上のフォルダにある場合(親ディレクトリ)
::text.txt

[4]絶対位置
MacHDD:text.txt

アプリケーションのなかには最初の起動時に絶対位置を取得してプロテクト状態にしてしまうようなものもあります(Lightwave 3D ver.5.x, Page Maker ver 4.x)。Macintoshの場合，Windowsなどと異なりハードディスクの名前を変更することができます。そのため通常のプログラムを作成する場合[4]の絶対パス指定は，ほとんど使われません。また，フォルダ名，ファイル名，:(コロン)をあわせて255文字以内で指定するという制限があるので絶対パスでは深い階層のファイルまで指定することは難しくなります。

ファイルの位置指定部分を組み込んだのがsample2.basです。コメントになっている部分を元に戻して実際にどのファイルが参照されるか確認してみてください。

■ユーザーにファイルを指定させる

いままでは「すでにわかっているファイル」を指定して読み込み表示しました。初期設定ファイルやアプリケーションが独自に使用するファイルであれば問題ありません。しかしユーザーが指定したファイルを読み込むという状況のほうが圧倒的に多いはずです。となると，なんらかの方法でユーザーにファイル名を指定してもらう必要があります。BASICであれば昔ながらの，

INPUT "ファイル名",filename$

というINPUT文を使った方法も使えます。Future BASICも一応上記のような方法が使えます。しかし上記の方法ではユーザーはファイルのパスを指定しなければなりません。困るのは，ファイルがどこにあったか思い出せない，ファイル名を忘れてしまっている場合です。これでは困ります。

Macintoshでは図2のようなファイル選択ダイアログ[*1]を表示してユーザーにファイルを選択してもらうのが一般的です。このダイアログを簡単に表示でき，ファイル選択までの処理を自動的に行ってくれれば非常に便利です。実はこのサービスはOS側で行われるため，わずか1命令で実現することができます。その命令はFILES$でファイル選択ダイアログの場合，以下の書式になります。

文字列変数名 = FILES$(_fOpen,ファイルタイプ,,ボリューム参

照番号)
(例：filename$ = FILES$(_fOpen,"TEXT",,volRefNum%))

ユーザーがファイル名を指定した場合，文字列変数名とボリューム参照番号にファイル名，値が格納されます。もしファイルが選択されずにキャンセルされた場合はファイル名は空になります。

ファイルタイプについて説明します。ファイルタイプは「ファイルの種類」を表しています。ファイルタイプは4文字で構成されておりTEXTならTEXT，PICTならPICT画像，MooVならQuickTimeムービーという具合に決められています(表1参照)。

ファイルタイプを指定するとダイアログには該当するファイル以外は表示されません*5。

もし複数のファイルタイプを指定する場合は列記します。TEXTとPICT形式のデータを読み込みたいのであれば以下のようになります。

filename$ = FILES$(_fOpen,"TEXTPICT",,volRefNum%)

逆にすべてのファイルを読み込みたい場合はなにも指定しません。つまり以下のように指定します。

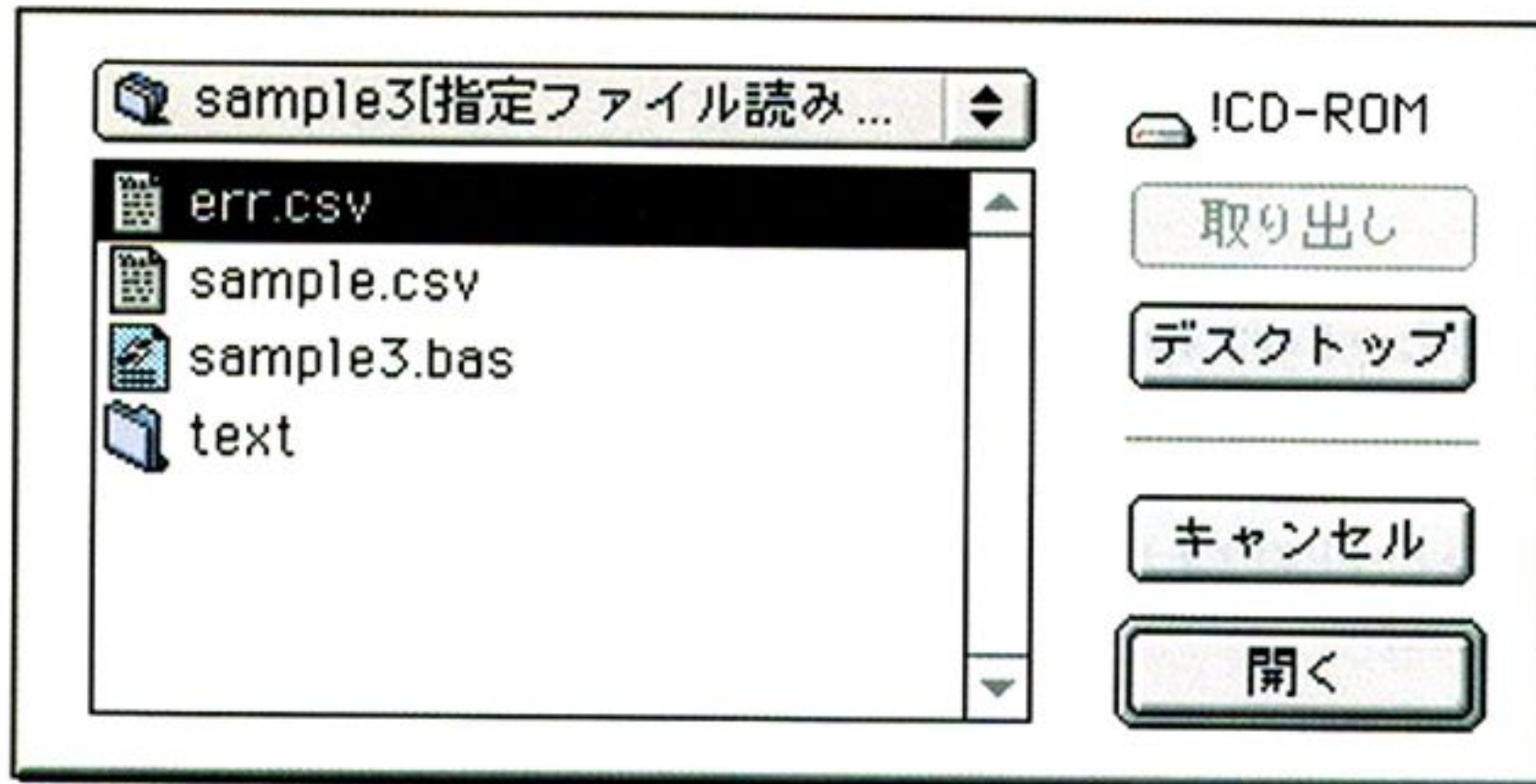



図2　ファイル選択ダイアログ。ここでファイル名を選択する

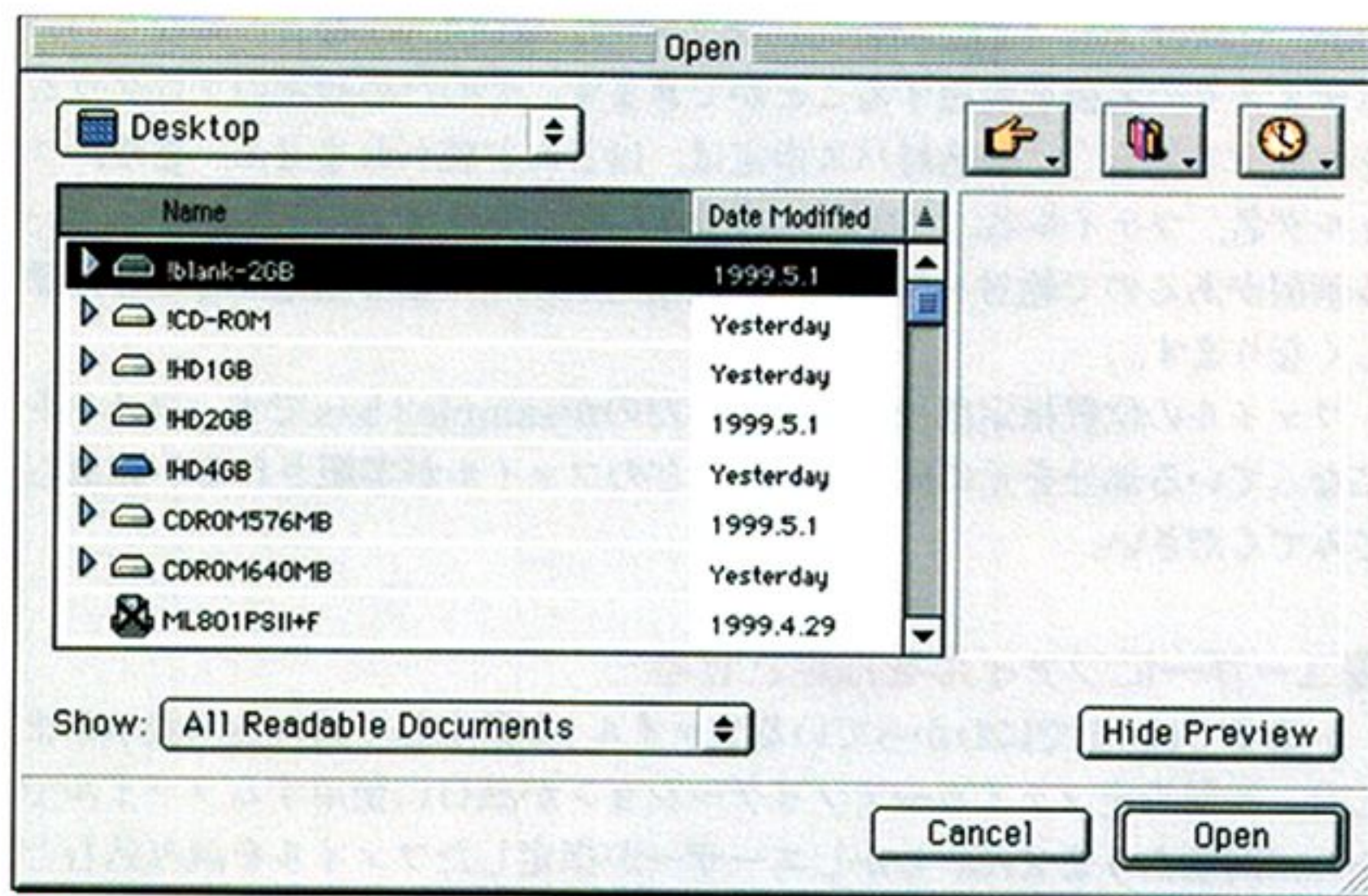



図3　MacOS8.0からサポートされた新しいファイル選択ウィンドウ。表示サイズを変更できる

リスト3　sample3.bas

```
'-----------------------------------------------------
' "テキストデータを読み込んでウィンドウに表示する"
'-----------------------------------------------------
CALL GETFNUM("Osaka",fontID%)
TEXT fontID%
filename$ = FILES$(_fOpen,"TEXT",,volRefNum%)
OPEN "I",#1,filename$,,volRefNum%
WHILE NOT EOF(1)
  INPUT #1,txt$
  PRINT txt$
WEND
CLOSE #1

BEEP
DO
  HANDLEEVENTS
UNTIL _false
```

表1

■クリエーター

クリエーター	アプリケーション
<SL>	Silverlining 5.6
8BIM	Adobe Photoshop 3.0J
aCa2	Font Grapher 3.3
AD3D	Adobe Dimensions 2.0
AFTB	Afterburner
AI5J	Adobe Illustrator 5.5
ALD4	Aldus Page Maker 4.5
ALK5	Aldus Page Maker 5.0
ALK6	Adobe Page Maker 6.0
ART3	Adobe Illustrator 3.0
ART5	Adobe Illustrator 5.0/7.0
ARTY	Adobe Illustrator 1.1
ARTZ	Adobe Illustrator 88
ASFL	Font Downloader
BABL	DeBabelizer
BEdt	Bin Edit
BKCT	Magic Works Ver 2.0
Bry2	KPT Bryce 2
Bry3	Bryce 3D
BsCW	B's Crew
BSRE	B'sRecorder Pro 4.0
CARO	Acrobat Reader 3.0/Exchange
cNif	ComNifty
CPCT	Compact Pro
CSOm	Eudora 1.3.8/Pro
CWIE	Code Warrior (IDE)
dPro	MacDraw Pro
DSTL	Acrobat Distiller 3.0J
Edt7	Edit 7
EWd3	KPT Bryce
FMJ3	File Maker Pro
FTCh	Fetch
FXTC	Adobe After Effects 3.1
gfBr	GIF Builder
GKON	Graphic Converter
GoMk	GoLive CyberStudio 3 (Adobe GoLive)
GSBM	Drive 7
jB·x	Sound Edit 16
JEDT	Jedit 1/2
LARC	MacLHa
LVIn	LogoVista Internet
LW3D	LightWave 3D Ver 5.0
LWMD	LightWave 3D Ver 5.0 (モデラー)
MAF1	Aftershock
McPL	Mac Perl
MD93	MacroMedia Director 4
MD95	MacroMedia Director 5
MD97	Macromedia Director 6/6.5

```
filename$ = FILES$(_fOpen,"",,volRefNum%)
```

FILES$で取得したファイル名とボリューム参照番号をOPEN命令で指定すればユーザーが指定したファイルを開くことができます。上記の場合は以下のようにOPEN命令でファイル名とボリューム参照番号を指定します。

```
OPEN "I",#1,filename$,,volRefNum%
```

実際のプログラムはsample3.basのようになります。

＊1　現在は最新のナビゲーションサービスによって図3のような，より高度なファイル選択を行うことも可能です。
＊2　Macintosh Easy Open/Mac OS Easy Openコントロールパネルの書類のスイッチがオンになっている場合はファイルタイプに関係なく表示されてしまいます。このためFuture BASICでは不具合が発生してしまうことがあります。逆にGIF,JPEGなども自動的に変換して描画してくれるので便利といえば便利です。諸刃の剣みたいな感じです。

テキストファイルの書き出し

テキストファイルの読み込みはできましたので今度はテキストファイルの

MFL2：MacroMedia Flash 2/3
MGIC：Color Magician III
MKBY：Macromedia Fireworks
MMCR：六角大王 Super
MMDR：MacroMedia Director 3
MMKE：ミミカキエディット
MOSS：Netscape Navigator 1.1～4.5
mPAK：Mpack 1.5J
MRIP：TScript 4.0
MSIE：Internet Explorer 2～4.5
MSWD：Microsoft Word
NAVm：Norton AntiVirus
NCSA：NCSA Telnet
P3DA：Painter 3D
PNnu：Norton Utility
PrMr：Adobe Premiere 4.0
PZ3A：Poser 3
Roku：六角大王 Ver 5.5
RSED：Res Edit
SAnP：Scenary Animator 1.2
SCS!：SCSI Probe
SI-D：Metacreations Infini-D ver 4.0/4.5
SIT!：StuffIt Deluxe
SITx：StuffIt Expander
StMl：Adobe PageMill
TrAn：Transit
ttro：TeachText,SimpleText読み込み専用
ttxt：SimpleText
ttxt：TeachText
TVOD：Movie Player
viOj：Poser Ver 2.0
VIPR：Video Monitor Pro
VisP：Vista pro 1/2
VShp：Strata VideoShop
WMLR：Will Mail
Wrap：ShrinkWrap
XCEL：Microsoft Excel Ver 5～98
XPRa：Quark Xpress 3.1
xxxX：Shade III Light 1.2.5
ZBAS：Future BASIC 1.0x/II

■ファイルタイプ

..CT：サイテックスCT画像
????：タイプ不明
3DMF：QuickDraw 3D Meta File
8BIM：Photoshop 画像
aCf2：Font Grapher書類
AD3D：Adobe Dimension書類
AIFC：AIFF圧縮サウンド
AIFF：AIFFサウンド
ALB2：Aldus Page Maker 2.0J書類
ALB3：Aldus Page Maker 3.0J書類
ALB4：Aldus Page Maker 4.5J書類
ALB5：Aldus Page Maker 5.0J書類
ALB6：Adobe Page Maker 6.0J書類
APPL：アプリケーション
BINA：MS-DOSバイナリ形式
BMPp：BMP形式
cdev：コントロールパネル書類
dDoc：MacDraw形式
DEXE：Windowsアプリケーション(実行ファイル)
dImg：ディスクイメージ
EPSF：EPSF画像
Film：Film Strip形式
GIFf：GIF画像
HELP：Helpファイル
ILBM：IFF形式
jB1 ：Sound Edit 16サウンド
JPEG：JPEG画像
LARC：LHarc圧縮形式
LHA ：LHa圧縮形式
Midi：MIDI形式
MooV：QuickTimeムービー
MPG：MPEGファイル
MV93：MacroMedia Director 4.0ムービー
MV95：MacroMedia Director 5.0ムービー
NSPL：Netscapeプラグイン
PACT：CompactPro圧縮形式
PICS：PICS形式(連番PICT)
PICT：PICT画像
PNTG：Mac Paint画像
PREF：プリファレンス
pref：プリファレンス
PXR ：Pixar画像
rsrc：リソースデータ
SAnF：Scenary Animator書類
SCRN：スタートアップスクリーン
SIT!：StuffIt 1.5.1圧縮形式
SIT2：StuffIt Deluxe圧縮形式
TEXT：テキスト形式
TIFF：TIFF画像
TPIC：PIC形式（X68000とは異なる）
TPIC：TGA画像
ttro：TeachText Read Only形式
VWMD：MacroMedia Director 3.0ムービー
WAVE：WAVEサウンド
XDOC：Quark Xpress 文書

図4　せっかく作ったのにアイコン真っ白。おまけに開くこともできない

図5　今度は正しいアイコンになった。もちろんテキストエディタで開くことができる

リスト4　sample4.bas

```
'----------------------------------------------------
' "テキストデータをファイルに出力する"
'----------------------------------------------------
DEF OPEN "TEXTttxt"
OPEN "O",#1,"out-file"
PRINT #1,"ファイルを出力してみます"
PRINT #1,"日本語も問題なく出力できます"
CLOSE #1
```

書き出しです。ファイルにテキストを書き出すにはPRINT文を使います。これは普通のPRINT文とまったく同じです。違うのはファイルに出力するためにファイル番号を付加しなければならない点です。たとえば"Oh!mz"の文字をファイル番号1に出力するには以下のように記述します。

```
PRINT #1,"Oh!mz"
```

英語だけでなく日本語も問題なく出力できます。出力する場合はOPEN命令もファイルモードを入力専用の"I"でなく出力専用の"O"にする必要があります。実際は以下のようになります。

```
OPEN "O",#1,"out-file"
PRINT #1,"Oh!mz"
CLOSE #1
```

ところがこのプログラムを実行して実際にできるファイルを見るとアイコンが真っ白。おまけにテキストファイルなのにテキストエディタ(SimpleTextなど)で開くこともできません(図4)。

なぜ，こんなことになってしまったのかというと「ファイルタイプを指定していない」ためです。ファイルタイプを指定するにはDEF OPEN命令を使います。書式は以下のようになります。

```
DEF OPEN "ttttcccc"
(tttt:4文字のファイルタイプ，cccc:4文字のクリエータ)
```

ファイルタイプとクリエータについては表1を参照してください。標準で入っているテキストエディタであるSimpleTextの書類にして保存するにはファイルタイプをTEXT，クリエータをttxtとします。

```
DEF OPEN "TEXTttxt"
```

このように記述すればテキスト書類となりSimpleText書類のアイコンになります(図5)。

ただしDEF OPEN命令はOPEN命令の前に記述しないと意味がありませんので注意してください。実際のプログラムはsample4.basのようになります。

ユーザーに保存先を指定してもらうには

sample4.basはアプリケーションと同じフォルダにファイルを作成します。ファイルを読み込んだときと同様にパスを指定すれば異なるフォルダにも保存することができます。しかし，ファイルの保存先はユーザーに指定してもらう場合がほとんどです。図6のようなファイル保存ダイアログを表示しファイル名を入力してもらうことになります。

ファイル選択ダイアログが簡単に呼び出せたように，このファイル保存ダイアログも1命令で簡単に使用することができます。その命令はファイル選択ダイアログ同様FILES$です。書式は以下のようになります。

```
filename$=FILES$(_fSave,プロンプト，暫定ファイル名，ボリューム参照番号)
```

プロンプト，暫定ファイル名は図7に示す部分の表示文字です。ボリューム参照番号は変数を指定します。これはユーザーが保存ファイル名を指定した場合に，どの場所のファイルかを示すためにボリューム参照番号が格納されます。このボリューム参照番号をもとにOPEN命令でファイルを開きます。これはファイルを読み込むのとまったく同じです。

実際のプログラムはsample5.basのようになります。

ユーザーがキャンセルボタンを押してしまった場合は返されるファイル名は空になります。サンプルプログラムでは，そのようなエラー処理を省いています。正しく処理するならば，

```
filename$=FILES$(_fSave,"ファイル名：","名称未設定
```

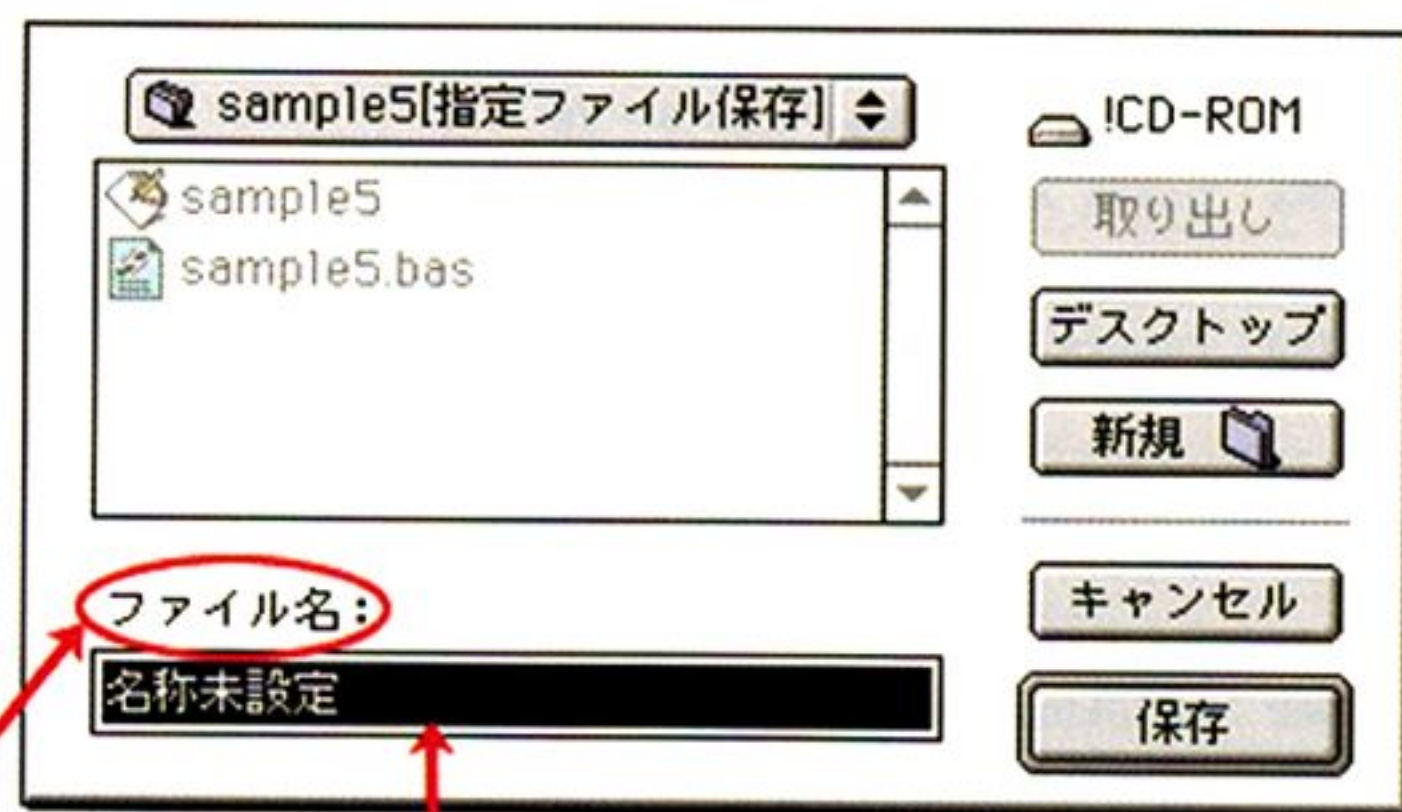
図6　ファイル保存ダイアログ。MacOS 8.0以降ではナビゲーションサービスによる保存ウィンドウもある

プロンプト
暫定ファイル名

図7　矢印で示した部分をFILES$命令で指定する

```
  ",volRefNum%)
  LONG IF filename$<>""
    OPEN "O",#1,filename$,volRefNum%
    PRINT #1,"ファイルを出力してみます"
    PRINT #1,"日本語も問題なく出力できます"
    PRINT #1,"ファイル名も指定できます"
    CLOSE #1
  END IF
```

のようになります。

ファイルコピー

今度はファイルのコピープログラムを作ってみます。コピーといってもMacintoshにはデータだけでなくリソースもあって，本当はリソースもコピーしなければいけないのですが，ここでは単純にデータのみのコピーを行います(一種のリソースカッター)。またファイルタイプは強制的にテキストにしているためファイルタイプやクリエータまで完全にコピーするわけではありません。

ファイルのコピーといってもいままでの方法と同じです。異なるのはデータの読み込みと書き込みの部分です。INPUT #では1行が256文字までという制限があり，さらに改行コードが存在せず行末がEOF(ファイルエンド)になっている場合はエラーになってしまいます。このような場合はINPUT #, PRINT #ではなく別の命令を使います。それはREAD #，WRITE#です。それぞれ読み込み，書き込みを行います。

```
READ #, WRITE #は以下の書式になります。
  READ #ファイル番号,変数名1,変数名2….,変数名n
  READ #ファイル番号,変数名;読み込みサイズ
 (例：READ #1,ohx
       READ #1,ohmz$;2)

  WRITE #ファイル番号,,変数名1,変数名2….,変数名n
  WRITE #ファイル番号,変数名;書き込みサイズ
 (例：WRITE #1,ohx
       WRITE #1,ohmz$;2)
```

＊変数名2以降は省略可能

読み込み/書き込みサイズはバイト単位になります。ここでは1バイト読んで，1バイト書き出すというもっとも効率の悪い方法でファイルのコピーを行います。効率は最悪ですがプログラムはシンプルです。実際のプログラムはsample6.basのようになります。

最後に

sample1～sample6までのプログラムには終了メニューがありませんので「コマンド+.(ピリオド)」キーを同時に押して終了させてください。

INPUT #を使った場合，Future BASICの都合で注意しなければならないことがあります。文字列の末尾が改行でなくEOF(ファイルエンド)で終わっている場合正しく読み込むことができずにランタイムエラーになってしまいます。ほとんどバグみたいなものですが，これで困ってしまうこともあります。次のバージョンでは修正されると思いますが，イマイチよろしくない状態です。CSV形式などに変換する場合には注意しましょう。

あとエラー処理は行っていませんので各自本文を参照してエラー処理を付け加えてみてください。

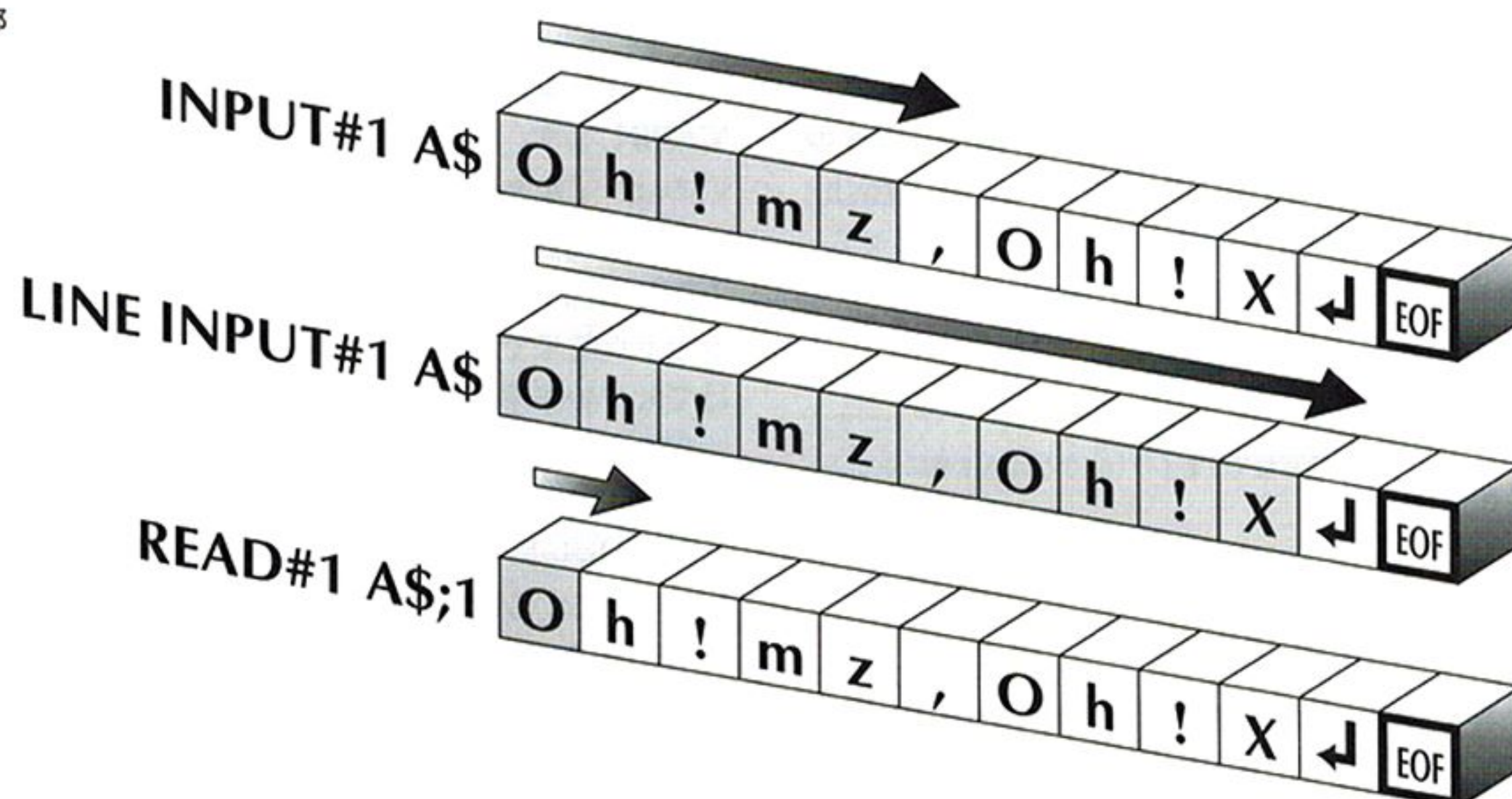
sample6.basでは1バイトずつ読み書きしていますが，サイズの大きいファイルでは致命的に遅くて使えません。それにはちゃんとした命令が用意されています。次回はファイルの高速読み書きとメモリ関係について説明します。メモリ関係はちょっと面倒なんですが避けて通れません。では，また次回。

リスト5　sample5.bas

```
'--------------------------------------------------------
' "テキストデータをファイルに出力する"
'--------------------------------------------------------
DEF OPEN "TEXTttxt"
filename$=FILES$(_fSave,"ファイル名：","名称未設定",volRefNum%)
OPEN "O",#1,filename$,volRefNum%
PRINT #1,"ファイルを出力してみます"
PRINT #1,"日本語も問題なく出力できます"
PRINT #1,"ファイル名も指定できます"
CLOSE #1
```

リスト6　sample6.bas

```
'--------------------------------------------------------
' "1バイト読み込み＆書き込み"
'--------------------------------------------------------
DEF OPEN "TEXTttxt"
filename$ = FILES$(_fOpen,"TEXT",,volRefNum%)
savename$=FILES$(_fSave,"ファイル名：","名称未設定",volRefNum2%)
OPEN "I",#1,filename$,volRefNum%
OPEN "O",#2,savename$,volRefNum2%
WHILE NOT EOF(1)
  READ #1,rd$;1
  WRITE #2,rd$;1
WEND
CLOSE #1,#2
```

図8　今回説明した命令のファイルからの読み込み範囲。グレーの部分が一度に読み込まれる範囲

JavaScriptから始めるプログラミング　第3回
IE 4/5のフィルタ効果を使ってデモを作る

古籏一浩　KaZuhiro FuRuhata

今号は2次元系グラフィックということでJavaScript講座もちょっとだけ画像寄り(?)なネタです。今回は残念ながらWindows版のInternet Explorer 4/5でしか動作しません。同じExplorerでもMac, UNIX版では動作しませんので，あしからずご了承ください。

JavaScriptを使ったデモ

JavaScriptを使って映像デモを作成してみます。映像デモ……まあAMIGAでいうところのメガデモといったところです。ただし今回は映像のみということでサウンドはありません。Web上でぐりぐり動かすのであればJavaScriptなんか使うよりもMPEG/QTムービーもしくはMM Flashを使うほうがはるかに簡単かつ見栄えがよいものができます。サウンドも同期ですし，制限がほとんどありません(容量=見るまでにかかる時間が最大の制限でしょうか)。

「制限の少ないものよりも制限だらけのほうが燃える」(?)ということで制限の多いJavaScriptでデモを作ってみましょう。

制限が多いと書きましたが，Netscape NavigatorとInternetExplorer(以下，Explorer)双方で動作させるとなると実際のところほとんどなにもできない状態になってしまいます。そこで開き直ってフィルタ効果などが用意されているExplorer 4以降で作成することにします。Explorer 4用で作成するとなると以下のことが実現可能です(ただしプラグインが必要なものを除く)。

(1)レイヤーの表示/非表示*0
(2)レイヤー位置の変更*0
(3)レイヤーサイズ/表示サイズの変更*0
(4)文字サイズ/フォントの変更*0
(5)画像サイズ/画像の変更*0
(6)2Dベクトルグラフィックの制御
(7)フィルタ効果

これらを組み合わせてデモを作成します。(1)～(5)に関してはOh!X vol.1の本講座を参照してください。また(6)の2Dベクトルグラフィックの制御および(7)のフィルタ効果はMacintosh/UNIX版のExplorer 4/4.5では動作しません。今回の講座では残念ながら対象外です。

まずフィルタ効果について説明します。

*0
Netscapeでも実現可能ですが(4)(5)などは速度的につらいものがあります。

フィルタ効果

Internet Explorer 4から表示する文字，画像などに対していくつかのフィルタ(特殊)効果を施すことができるようになっています。使用できるフィルタ効果は以下のとおりです。

[1]グレー変換	[2]ネガ	[3]水平反転
[4]垂直反転	[5]白黒(反転グレー)	[6]マスク
[7]クロマキー合成	[8]ぼかし	[9]シャドウ
[10]発光	[11]α合成	[12]波形
[13]ライト効果	[14]切り替え	

これらのフィルタはタグ内のオプションで指定することができます。となるとJavaScriptの出番がないような気もします。が幸い(?)このフィルタ効果をJavaScriptで制御することができます。ただしフィルタ効果には結構仕様上のミスがあって期待どおりにいかないことがあります。[1]～[7][13][14]の効果には不具合はありませんが，それ以外のフィルタ効果には不具合があります。この不具合については最後に説明します。まずはタグレベルでフィルタ効果を確認してからJavaScriptで制御することにしましょう。

タグ内での指定

まずJavaScriptで制御する前にタグレベルでの指定方法です。タグのオプションとして指定する場合は，

```
filter : フィルタ名(フィルタオプション)
```

となります。注意しなければいけないのはスタイルシートを設定していない場合フィルタオプションを指定してもフィルタ効果は反映されない点です。

最低限のスタイルシート設定は以下のようになります。

```
STYLE="position:absolute"
```

このオプションとフィルタオプションを併記して使用します。もうひとつ注意しなければいけないのはフィルタ効果が反映されるのはスタイルシートで設定された矩形範囲内でしか表示されずはみだした部分はクリップされてしまうことです。シャドウ効果であれば影を大きくずらすと影が欠けてしまいます。スタイルシートで横幅/縦幅を設定しない場合は文字/画像のサイズから自動的に設定されます。横幅/縦幅を指定するにはwidth，heightオプションを使います。横幅300，縦幅200にするには以下のように指定します。

```
STYLE"position:absolue;width=300;
height=200;"
```

次にフィルタを指定します。影をつけるフィルタであるshadowの場合以下のようになります。

```
filter:shadow(color=blue,direction=1
35)
```

colorは影の色，directionは影の角度を示します。カッコ内にフィルタオプションを記述しますがなにも記述しなくても構いません。なにもオプションが記述されていない場合はデフォルト値が使用されます。

各フィルタのオプションを表1に示します。また実際のプログラムはsample1.html～sample19.htmlのようになります。ライト効果とワイプ効果のみJavaScriptを使用する必要があります。

JavaScriptで制御するには

JavaScriptでフィルタ効果を制御するには以下のように指定します*1。

```
オブジェクト名.filters[〝フィルタ名〞].プ
ロパティ名 = 設定値
```

オブジェクト名ですがNAMEオプションでつけたものではなくIDオプションで指定した名前になります。フィルタ名は表1に示すものを指定します。myObjという名前のオブジェクトのdropShadowフィルタ効果を指定するには以下のようになります。

```
myObj.filters[〝dropShadow〞]
```

このオブジェクト/フィルタの設定値，影の色を青色にするには以下のようになります。

```
myObj.filters[〝dropShadow〞].color
= 0x0000FF
```

blueなどカラー名を指定すると型が違うというエラーになってしまいます。#0000FFという指定もエラーになります。

フィルタ効果を解除するにはenbaledプロパティにfalseを代入します。再度効果をかけるにはtrueを設定します。具体的には以下のように指定します。

・フィルタ効果解除

```
myObj.filters
[〝dropShadow〞].enabled = false
```

・フィルタ効果適用

```
myObj.filters
[〝dropShadow〞].enabled = true
```

プロパティは設定だけでなく読み出すこともできます。注意しなければならないのはタグで対応したフィルタ効果が設定されていないとエラーになってしまう点です。

●シャドウフィルタ

```
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:shadow(color=blue,direction=135)">
<H1>シャドウを付けてみました。</H1>
</DIV>
```

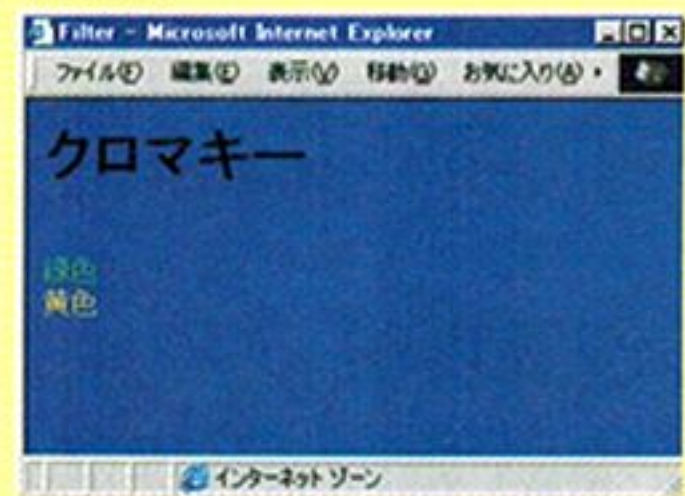
●α合成フィルタ

```
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:alpha(opacity=0,finishOpacity=100,
style=1)">
<H1>α合成してみました</H1>
</DIV>
```

●blur効果フィルタ

```
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:blur(direction=45,strength=7)">
<H1>blur効果です</H1>
</DIV>
```

●クロマキーフィルタ

```
<BODY bgColor="#0000FF">
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:chroma(color=#FF0000)">
<H1>クロマキー</H1>
<FONT COLOR="#FF0000">赤色</A><BR>
<FONT COLOR="#00FF00">緑色</A><BR>
<FONT COLOR="#FFFF00">黄色</A><BR>
</DIV>
</BODY>
```

●ドロップシャドウフィルタ

```
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:dropShadow(color=#9090E0)">
<H1>ドロップシャドウ</H1>
</DIV>
```

●水平反転フィルタ

```
<DIV
STYLE="position:absolute;
width=150;height=100;
filter:flipH()">
<H1>水平反転</H1>
</DIV>
```

●垂直反転フィルタ

```
<DIV
STYLE="position:absolute;
width=150;height=60;
filter:flipV()">
<H1>垂直反転</H1>
</DIV>
```

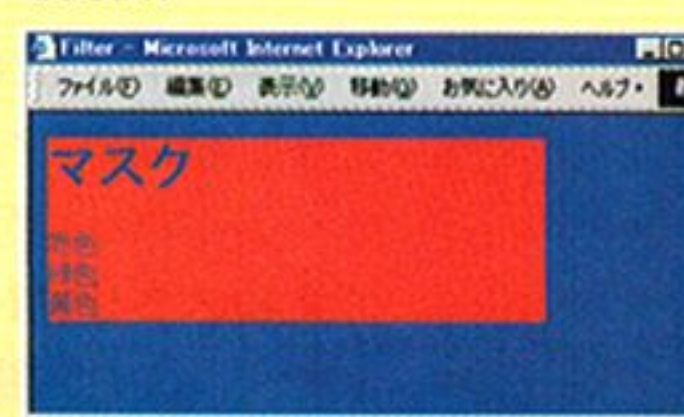
●発光フィルタ

```
<BODY bgColor=black>
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:glow(color=#FFFF00,strength=6)">
<H1>発光効果</H1>
</DIV>
</BODY>
```

●グレーフィルタ

```
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:gray()">
<H1>グレー効果</H1>
<IMG SRC="image.jpg">
</DIV>
```

●マスクフィルタ

```
<BODY bgColor="#0000FF">
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:mask(color=#FF0000)">
<H1>赤色でマスク</H1>
</DIV>
</BODY>
```

●ネガフィルタ

```
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:invert()">
<H1>ネガ効果</H1>
<IMG SRC="image.jpg">
</DIV>
```

●シャドウフィルタ

```
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:shadow(color=#9090E0)">
<H1>シャドウ</H1>
</DIV>
```

●波フィルタ

```
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:wave(freq=4,strength=10)">
<H1>波効果</H1>
<IMG SRC="image.jpg">
</DIV>
```

●白黒フィルタ

```
<DIV
STYLE="position:absolute;
width=300;height=100;
filter:Xray()">
<H1>白黒</H1>
<IMG SRC="image.jpg">
</DIV>
```

●ブレンドフィルタ

```
<HEAD>
<SCRIPT Language="JavaScript">
<!--
function change1()
{
        myIMG.filters["blendTrans"].Apply();
        myIMG.src = "image.jpg";
        myIMG.filters["blendTrans"].Play();
}
function change2()

{
        myIMG.filters["blendTrans"].Apply();

        myIMG.src = "image2.jpg";
        myIMG.filters["blendTrans"].Play();
}
// -->
</SCRIPT>
</HEAD>
<BODY>
<H1>ブレンド効果</H1>
<IMG SRC="image.jpg" ID="myIMG"
STYLE="position:absolute;
filter:blendTrans(duration=4)">
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<BR><BR><BR><BR>
<FORM STYLE="position:absolute;top=300">
<INPUT TYPE="button" VALUE="image1" onClick="change1()">
<INPUT TYPE="button" VALUE="image2" onClick="change2()">
</FORM>
</BODY>
```

●ワイプフィルタ

```
<HEAD>
<SCRIPT Language="JavaScript">
<!--
function change1()
{
        myIMG.filters["revealTrans"].Apply();
        myIMG.src = "image.jpg";
        myIMG.filters["revealTrans"].Play();
}
function change2()
{

        myIMG.filters["revealTrans"].Apply();
        myIMG.src = "image2.jpg";
        myIMG.filters["revealTrans"].Play();
}
// -->
</SCRIPT>
</HEAD>
<BODY>
<H1>ワイプ効果</H1>
<IMG SRC="image.jpg" ID="myIMG"
STYLE="position:absolute;
filter:revealTrans(duration=2,Transision=10)">
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<BR><BR><BR><BR>
<FORM STYLE="position:absolute;top=300">
<INPUT TYPE="button" VALUE="image1" onClick="change1()">
<INPUT TYPE="button" VALUE="image2" onClick="change2()">
</FORM>
</BODY>
```

●ライトフィルタ/環境光

```
<HEAD>
<SCRIPT Language="JavaScript">
<!--
function setLight()
{

myIMG.filters["Light"].addAmbient(255,255,255,70);
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="setLight()">
<H1>ライト効果（環境光）</H1>
<IMG SRC="image.jpg" ID="myIMG" STYLE="position:absolute;
filter:Light;">
<BR>
</DIV>
</BODY>
```

●ライトフィルタ/スポットライト

```
<HEAD>
<SCRIPT Language="JavaScript">
<!--
function setLight()
{

myIMG.filters["Light"].addCone(70,40,80,160,120,255,255,2
55,80,100);
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="setLight()">
<H1>ライト効果（スポットライト）</H1>
<IMG SRC="image.jpg" ID="myIMG" STYLE="position:absolute;
filter:Light;">
<BR>
</DIV>
</BODY>
```

●スポットライトが移動する。でもちょっと変

```
<HEAD>
<SCRIPT Language="JavaScript">
<!--
function newLight()
{

myIMG.filters["Light"].addCone(160,120,10,0,0,100,100,255,80,100);
}
function setLight()
{
        x = event.x;
        y = event.y;
        myIMG.filters["Light"].MoveLight(0,x,y,10,true);
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="newLight()" onMousemove="setLight()">
<H1>ライト効果（スポットライト）</H1>
<IMG SRC="image.jpg" ID="myIMG" STYLE="position:absolute;
filter:Light;">
<BR>
</DIV>
</BODY
```

●ライトフィルタ/点光源

```
<HEAD>
<SCRIPT Language="JavaScript">
<!--
function setLight()
{

myIMG.filters["Light"].addPoint(160,40,30,0,255,255,100);
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="setLight()">
<H1>ライト効果（点光源）</H1>
<IMG SRC="image.jpg" ID="myIMG" STYLE=
                              "position:absolute;
filter:Light;">
<BR>
</DIV>
</BODY>
```

●なんとなくラスタスクロールしている感じ

```
<HEAD>
<SCRIPT Language="JavaScript">
<!--
function raster()
{
        myIMG.filters["wave"].strength -= 2;
        myIMG.filters["wave"].phase    += 2;
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="setInterval('raster()',10)">
<DIV ID="myIMG"
STYLE="position:absolute;
width=300;height=100;
filter:wave(freq=2,strength=200,lightstrength=60)">
<H1>波効果</H1>
<IMG SRC="image.jpg">
</DIV>
</BODY>
```

それでは，ちょっと面白そうな効果をやってみます。ひとつは昔流行った(？)ラスタスクロールもどき，もうひとつはスポットライトを使った効果です。まずラスタスクロールもどきです。これは徐々に振幅を小さくしていきます。タイマと組み合わせれば割と簡単に実現できます。sample20.htmlが実際のプログラムです。タグのオプションで振幅などの初期値を設定しておきます。あとはプロパティの値を減算/加算するだけです。

次にスポットライト効果です。これはライトがマウスのほうを照らす感じになります。まずページが読み込まれたらスポットライトを設定します。ページが読み込まれたかどうかはonLoadイベントを利用します。問題はマウスの座標の取得方法です。これはBODYタグにonMouseoverイベントを設定するだけでマウスが移動するたびに指定したイベントハンドラ(関数)が呼び出されます。呼び出し先でマウス座標を取得します。マウス座標は，

event.x　(X座標)
event.y　(Y座標)

として取得することができます。

＊1
本文の書き方以外にも以下のような記述もできます。

```
document.al("オブジェクト名").filters["フィルタ名"]
document.all["オブジェクト名"].filters["フィルタ名"]
document.all("オブジェクト名").filters.フィルタ名
document.all["オブジェクト名"].filters.フィルタ名
```

フィルタ使用にあたっての注意事項

一部のフィルタには不具合があります。といっても実害はあまりありません。図1のようにフィルタ効果を施すオブジェクト(テキスト/画像)がウィンドウからはみ出ていると正しい効果になりません。これは本来のテキスト/画像に対してフィルタ効果をかけているのではなく，ウィンドウ上でクリッピングされた後のデータに対してフィルタ効果をかけているためです。クリッピングされても影響のないgray,Xrayなどはよいのですが，シャドウや波などは設定値(strengthなど)を大きくすると表示されるはずのテキストや画像が一部しか表示されません。あとα合成効果も再設定がうまくいかないといった不具合もあります。

デモ用素材の注意事項

まずデモを作る前に用意したほうがよいのは映像素材です。素材として使えるのは，

・JPEG/GIF画像
・アニメーションGIF画像
・ムービー(AVI, MPEG, QuickTime)

といったところです。画像を用意しJavaScriptで制御すると，イマイチ動きがよくありません。単純にいえばがっかりするほど速度が遅いのです。秒間30フレームなんて夢の夢。秒間10フレーム出れば十分だろうといったところです。このためスクリプトの「最適化」を行わなければなりません。また期待する速度が出るような画像の工夫をする必要があります。

画像に関してはデータの容量によって表示速度は左右されません。速度的に左右されるのは表示サイズです。大きく表示させると時間がかかってしまいますので，大きな画像を表示させるのは避けたほうが無難です。単純な１枚絵でなくGIFアニメーションの場合は事情が少し異なります。JavaScriptでレイヤーや映像を制御するよりも「GIFアニメーションが優先して表示」されます。ですからスムーズな動きを実現させたい部分はGIFアニメーションを使うのがベターです[*2]。

そしてGIFアニメーションの場合は表示サイズ以外にデータサイズによっても速度が変わります。ディスクキャッシュから読み込まなければならないような大きなサイズの画像は避けるべきです。

また表示に関係ない部分の画像の縁/枠などはカットし，とにかくコンパクトに画像を作成する必要があります。

＊2
ただし同じGIF画像を大量に並べて表示させると，うまくアニメーションしません。そのような場合は全体を覆う１枚のGIFアニメ画像として用意するほうがベターです。

JavaScript制御の際の注意事項

結論からいえば実行速度に関してはJava Script側で努力するよりも映像素材と表示サイズでカバーしたほうがはるかに効果的です。しかし，いろいろな動作を組み合わせて実行させるとプログラムの組み方によっては見た目に速度が違ってしまうことがあります。Explorer 4/5の場合じっくり検証したわけではありませんがゲーム制作，今回のデモ制作で試した限りでは「行数によって速度が変わる」ようです。中間変数や演算をカットし直接的な制御および論理演算を使って行数を削減します。

本当はこうなる
でも実際はこんな具合

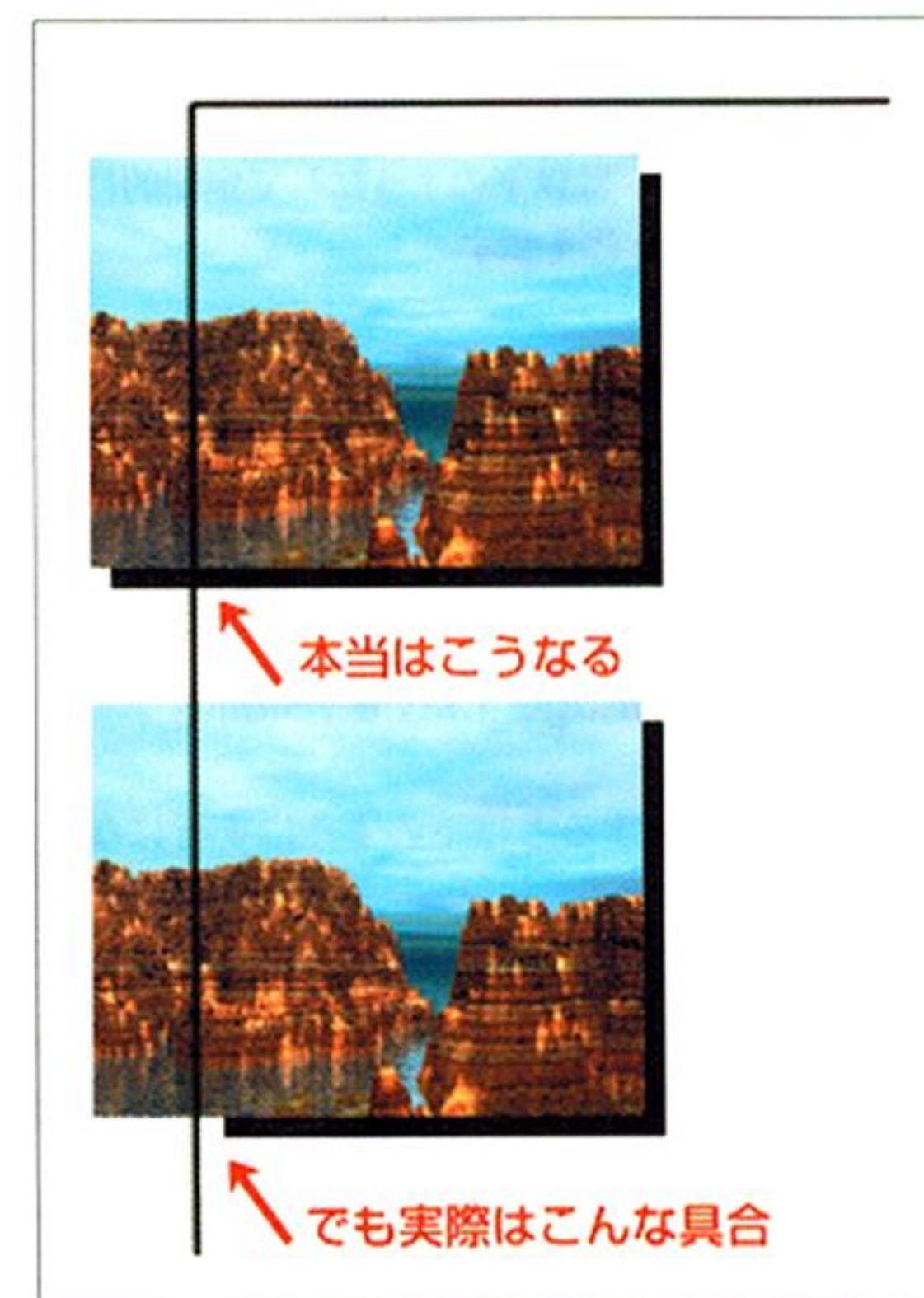



図1　フィルタ効果の不具合

表1　フィルタ効果オプション一覧

フィルタ名	メソッド/プロパティ名	設定値	内容
□alpha			
	enabled	trueまたはfalse	フィルタ効果スイッチtrue：オン。false：オフ
	finishOpacity	0～100	終了点の透明度。値が大きいほど不透明
	finishX	座標値	終了点のX座標
	finishY	座標値	終了点のY座標
	opacity	0～100	開始点の透明度。値が大きいほど不透明
	startX	座標値	開始点のX座標
	startY	座標値	開始点のY座標
	style	0～3	形状。0：フラット，1：ライン，2：放射状，3：四角
□blendTrans			
	Apply()	なし	設定
	duration	0以上	ワイプ切り替え秒数
	Play()	なし	再生開始
□blur			
	add	trueまたはfalse	false：元のイメージを合成しない。true：元のイメージを合成する
	direction	0以上45単位	角度。45度単位。¥
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
	strength	0以上	強さ
□chroma			
	color	#000000～#FFFFFF	抜く色
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
□dropShadow			
	color	#000000～#FFFFFF	影の色
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
	offX	オフセット	影のX方向のずれ具合
	offY	オフセット	影のX方向のずれ具合
	positive	trueまたはfalse	true：通常の影。false：カットアウト
□flipH			
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
□flipV			
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
□glow			
	color	#000000～#FFFFFF	発光色
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
	strength	0以上	強さ
□gray			
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
□invert			
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
□Light			
	addAmbient(r, g, b, str)		環境光設定
		r：0～255	赤
		g：0～255	緑
		b：0～255	青
		str：0～100	強さ
	addCone(x1, y1, z1, x2, y2, r, g, b, str, width)	スポットライト設定	
			x1：光源X座標　y1：光源Y座標　z1：光源Z座標
			x2：焦点X座標　y2：焦点Y座標
		r：0～255	赤
		g：0～255	緑
		b：0～255	青
		str：0以上	強さ
		width：0～90	幅(照射角度)
	addPoint(x, y, z, r, g, b, str)		点光源設定
			x：座標値　y：座標値　z：座標値
		r：0～255	赤
		g：0～255	緑
		b：0～255	青
	ChangeColor(No, r, g, b, type)		光源色変更
		No：光源番号	
		r：0～255	赤
		g：0～255	緑
		b：0～255	青
		type：trueまたはfalse	true：指定した色に変更。false：指定した色を加算
	ChangeStrength(No, str, tyep)		光源強度変更
		No：光源番号	
		r：0～255	赤
		g：0～255	緑
		b：0～255	青
		type：trueまたはfalse	true：指定した色に変更。false：指定した色を加算
	Clear()	なし	光源消去
	MoveLight(No, x, y, z, type)	光源移動	
		No：光源番号	
		x：座標値　y：座標値　z：座標値	
		type：trueまたはfalse	true：指定した座標に変更。false：指定した座標を加算
□mask			
	color	#000000～#FFFFFF	マスクカラー
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
□revealTrans			
	Apply()	なし	設定
	duration	0以上	ワイプ切り替え秒数
	Play()	なし	再生開始
	Transition	0～23	ワイプの種類(表2参照)
□shadow			
	color	#000000～#FFFFFF	影の色
	direction	0以上45単位	角度。45度単位
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
□wave			
	add	trueまたはfalse	false：元のイメージを合成する。true：元のイメージを合成しない
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ
	freq	0以上	波の数
	lightstrength	0～100	光源強度
	phase	0～100	位相
	strength	0以上	強さ
□Xray			
	enabled	trueまたはfalse	フィルタ効果スイッチ。true：オン。false：オフ

リスト

●demo.html（最適化されたもの）

```
<HTML>
<HEAD>
<TITLE>JS DEMO</TITLE>
<SCRIPT Language="JavaScript">
<!--
function demo()
{
        //  文字を上に移動させる
        myJSstr.style.top = (mojiY < -800) ? mojiY = 520 : mojiY -= 5;

        //  アルファチャンネルを制御
        myAlphaObj.opacity = alpha += alphaOffset;
        if ((alpha > 100) || (alpha < 0)) alphaOffset = -alphaOffset;

        //  タイトルをラスタースクロール
        myWaveObj.phase  += 6;

        //  文字を移動
        myKF.style.top  = (kfY += 4 * Math.sin((kfSin += 10) * 0.0174));
        myKF.style.left = (kfX > -240) ? kfX-- : kfX = 500;

        //  画面中に表示する
        myKageObj.visibility = ((multi++ & 0x7F) > 100) ? "visible" : "hidden";
}

//  デモ開始
function demoStart()
{
        mojiY = 200;                              //  JavaScript Demo文字のY座標
        alpha = 100;                              //  JavaScript Demo文字の透明度
        alphaOffset = -5;                         //  JavaScript Demo文字の透明度の積算数
        kfX = 500;                                //  1999 By K.F文字のX座標
        kfY = 280;                                //  1999 By K.F文字のY座標
        kfSin = 0;                                //  1999 By K.F文字の位相
        multi = 100;                              //  全画面影の表示カウンタ
        myKageObj  = myKage.style;                //  全画面影のオブジェクトを示す
        myWaveObj  = myWave.filters["wave"];      //  OpenSpaceのオブジェクトを示す
        myAlphaObj = myJSstr.filters["alpha"];    //  OpenSpaceのオブジェクトを示す
        setInterval("demo()",100);
}

// -->
</SCRIPT>
</HEAD>
<BODY onLoad="demoStart()">

<!--  背景  -->
<DIV ID="myBG" STYLE="position:absolute;top:0px;left:0px; width=480;">
<IMG SRC="bg.jpg" width="480" height="360">
</DIV>

<!--  タイトルロゴ  -->
<DIV ID="myWave" STYLE="position:absolute;top:60px;left:60px; width=480;z-index:90;
filter:wave(freq=2,strength=5,lightstrength=60)">
<IMG SRC="openspace.gif">
</DIV>

<!--  JavaScript Demoの文字  -->
<DIV ID="myJSstr" STYLE="position:absolute;top:200px;left:10px;width:120;z-index:200;
filter:alpha(opacity=100,finishOpacity=100,style=0)">
<IMG SRC="jsdemo.gif" width="120" height="670">
</DIV>

<!--  1999 by KaZuhiro FuRuhataの文字  -->
<DIV ID="myKF" STYLE="position:absolute;top:280px;left:500px;width:240;height:24px;z-index:120;
filter:alpha(opacity=80,style=0)">
<IMG SRC="kf.gif" width="240" height="24">
</DIV>

<!--  踊る怪しい人影  -->
<DIV ID="myGaikotsu" STYLE="position:absolute;top:80px;left:100px;width:240;height:24px;z-index:85;
filter:chroma(color=black)">
<IMG SRC="gaikotsu.gif" NAME="MAN" width="240" height="240">
</DIV>

<!--  踊る怪しい人影，全画面版  -->
<DIV ID="myKage" STYLE="position:absolute;top:0px;left:0px;width:480;height:360px;z-index:150;
filter:chroma(color=white)">
<IMG SRC="gaikotsu.gif" width="480" height="360">
</DIV>

</BODY>
</HTML>
```

実際のスクリプトはdemo.htmlです。最適化前のものがCD-ROMに収録されているdemo2.htmlです。工夫すると1/3程度にまで短くなるのがわかるでしょうか。その分だけ見にくくなり，わかりにくくなるのはしかたありません。

最適化について説明します。まず文字を上に移動させる部分のスクリプト部分です。最適化前のものは以下のようになります。

```
mojiY -= 5;
if (mojiY < -800) mojiY = 520;
document.all["myJSstr"].style.top =
mojiY;
```

上記のものを最適化すると以下のようになります。

```
myJSstr.style.top = (mojiY < -800) ?
mojiY = 520 : mojiY -= 5;
```

document.all［"myJSstr"］とmyJSstrはまったく同じものです。myJSstrがレイヤー名です。どちらもまったく同じ処理をしますが実行速度は異なります。直接レイヤー名を指定したほうが高速になります。document.all［"myJSstr"］ではドキュメントに含まれるオブジェクト中から毎回検索するため時間がかかってしまうのです。直接myJSstrを指定すれば検索は行われませんので高速に処理されます。同様にmyJSstr.filters［"alpha"］という指定も時間がかかるので一度変数に代入しておきプロパティのみ操作するようにすれば実行速度は上がります。

「オブジェクト名.style.top」とした場合と一度変数にオブジェクト名.styleまでを格納して「オブジェクト名.top」としてもほとんど動作速度は変わりません。

(mojiY < -800) ? mojiY = 520 : mojiY -= 5;は?前の条件式がtrueかfalseかによって処理を分けます。trueであれば?以降の文を実行しfalseであれば:以降の文を実行します。この場合はmojiYの値が-800以下になったらmojiYの値を520にし，そうでなければ5減算します。そしてここで代入された値がmyJSstr.style.topに格納されます。似たような手法でレイヤーの表示/非表示を切り替えることができます。時々画面に表示されるレイヤーは以下のようにして定期的に表示/非表示を切り替えます。

```
multi++;
if (multi > 128) multi = 0;
if (multi > 100) document.all
["myKage"] .
style.visibility = "visible"; else
document.all
        ["myKage"].style.visibility =
"hidden";
```

これを最適化すると以下のように1行になります。

```
myKageObj.visibility = ((multi++ &
0x7F) > 100) ? "visible" : "hidden";
```

multi++でmultiの値を1増やします。この値と0x7F（10進数で127）の論理積（AND）を取ります。0x7Fと論理積を取ると値は0～127の範囲に収まります[*3]。この値が100より大きいか比較し大きければレイヤーを表示する文字列

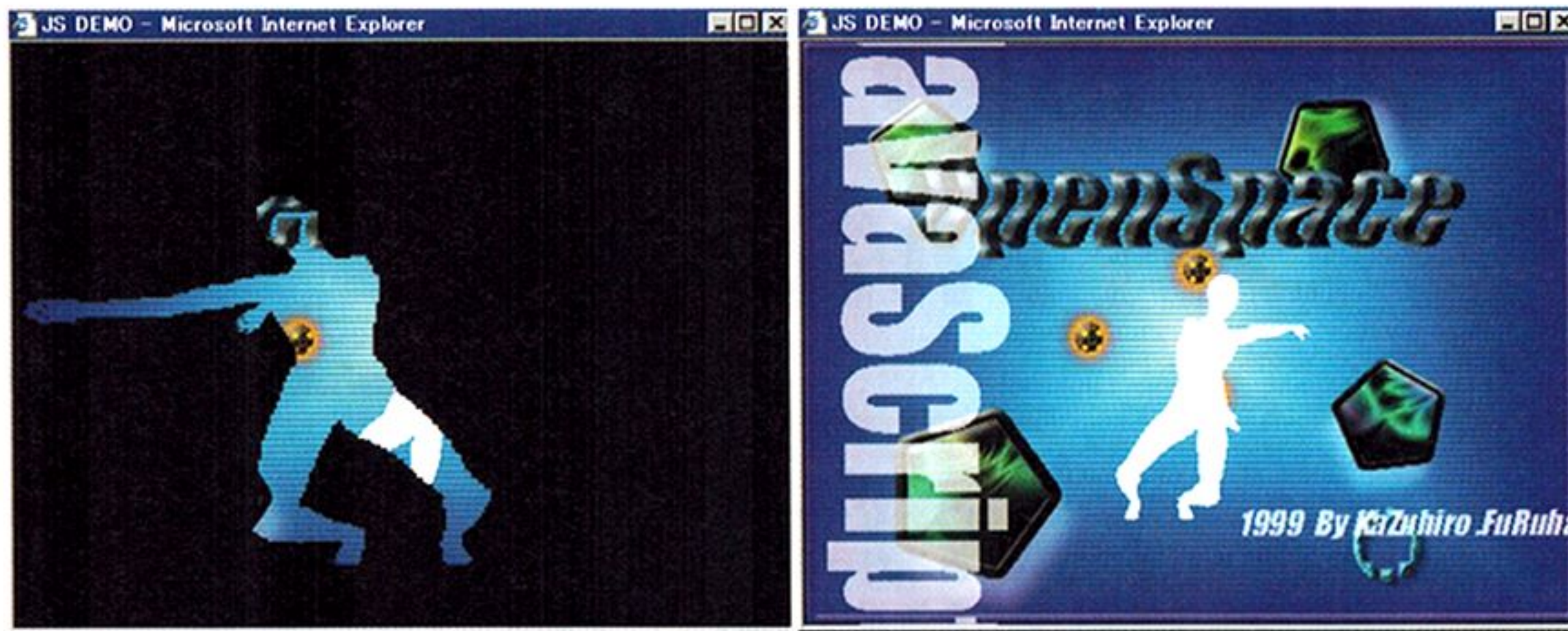



図2 一応デモ。サウンドがないのも難点

"visible", 100以下であればレイヤーを表示しない"hidden"をmyKageObj.visibilityに代入します。

ほかの部分も同様な作り方になっています。demo.htmlとdemo2.htmlを見比べて分解してみるとよいでしょう。

これでも実際のところ低速で実行されるのがオチです。そこで表示しない部分をクリッピングしたいところですが，クリッピング座標を指定すると自分自身に対してのローカル座標でクリッピングされてしまいます。そこでいちばんシンプルな逃げ道としてサブウィンドウを開いて指定サイズ，指定位置に表示しサブウィンドウ内に表示します。これでレイヤーを移動させたときに自動的にスクロールバーが表示されたり，クリッピング座標を設定することもなくなります。

しかしテキストを利用するとクリッピング領域(右側の座標)が画面右端になってしまいます(レイヤーに背景色を指定するとよくわかります)。これを防止するにはスタイルシートのwidthを使ってレイヤーの横幅を指定します。指定しなければいけないのは横幅だけで縦幅は指定しなくても大丈夫です。

実際の動作画面は図2のようになります。ちなみにサウンドをつけるのであれば<BGSOUND SRC="サウンドファイル名">とすればバックグランドミュージックとして流すことができます。

＊3 論理演算
論理積/論理和/排他的論理和は「ビット演算」を示します。通常の四則演算とは異なりビット同士を比較し一定のルールの下において結果を出力します。それぞれの論理演算のルールを以下に示します。

表2 ワイプ形状

設定値	ワイプ動作
0	四角(中央へ)
1	四角(外側へ)
2	円形(中央へ)
3	円形(外側へ)
4	下→上へ
5	上→下へ
6	左→右へ
7	右→左へ
8	ブラインド(縦)
9	ブラインド(横)
10	チェック模様(横)
11	チェック模様(縦)
12	ピクセルブラインド
13	両側から(左右→中央)
14	両側から(中央→左右)
15	両側から(上下→中央)
16	両側から(中央→上下)
17	斜めに(右上→左下)
18	斜めに(右下→左上)
19	斜めに(左上→右下)
20	斜めに(左下→右上)
21	ランダムライン(横)
22	ランダムライン(縦)
23	0～22のいずれか

最後に

プログラムの行数で競う方が面白いのかもしれません。またExplorer 5では，うまく動作しない場合があるかもしれません。危険度の高いものは往々にしてありがちということで。

次回は独自にオブジェクトを作る方法と作成したプログラムの再利用について説明したいと思います。作ったプログラムを再利用できるようにしておけば開発効率も上がり，デバッグに割く時間も減ります。では，また次回。

●論理積(AND)

「両方のビットが1のとき1，それ以外は0」

```
1 & 1 = 1
1 & 0 = 0
0 & 1 = 0
0 & 0 = 0
```

●論理和(OR)

「両方のビットが0のとき0，それ以外は1」

```
1 | 1 = 1
1 | 0 = 1
0 | 1 = 1
0 | 0 = 0
```

●排他的論理和(XOR)

「両方のビットが同じとき0，それ以外は1」

```
1 ^ 1 = 0
1 ^ 0 = 1
0 ^ 1 = 1
0 ^ 0 = 0
```

今回使用した論理積(AND)は両方のビットが1の場合1，それ以外は0になります。この法則を利用すると一定値以上の値にならないようにすることができます。例として本文の& 0x7F(127)と変数値との論理積を取ってみます。

```
10進数：2進数
6   = 00000110
65  = 00100001
127 = 01111111
130 = 10000010
255 = 11111111
```

5と127の論理積

```
  00000110(  6)
& 01111111(127)
---------------
  00000110(  6)
```

65と127の論理積

```
  00100001( 65)
& 01111111(127)
---------------
  00100001( 65)
```

130と127の論理積

```
  10000010(130)
& 01111111(127)
---------------
  00000010(  2)
```

255と127の論理積

```
  11111111(255)
& 01111111(127)
---------------
  01111111(127)
```

0から127まではそのままですが，それ以上の値では必ず127以下になります。

論理和は両方のビットが0のときに0，それ以外は1となります。論理和を利用すると常に値を奇数にできます。同様に2のn乗のゲタをはかせる(一定値を加える)こともできます。

・変数myValの値を奇数にする

```
myVal |= 1;
```

・変数myValの値に128を加える

```
myVal |= 0x80;
```

排他的論理和は両方のビットが同じ場合に0になります。排他的論理和を利用するとトグルスイッチ(オン/オフが切り替わる)が簡単にできます。その他，値を0にしたり(意味なし^^)，値を入れ替えたり，反転したデータを復元したりできます。

・トグルスイッチ。実行するたびにmyValの値が0，1交互に切り替わる。

```
myVal ^= 1;
```

・ゼロにする

```
myVal ^= myVal;
```

・myValとyouValを入れ替える

```
myVal ^= youVal; youVal ^= myVal;myVal ^= youVal;
```

・反転データを復元(グラフィック描画などに利用)

```
myVal ^= 0xFF; (データ反転)
 :
myVal ^= 0xFF; (データ復元)
```

論理演算を行う場合に注意しなければならない点があります。それは値は必ず整数でなければならないというものです。1.5など小数値などに対して論理演算を行うとNaN (Not a Number)となってしまいます。これを防ぐにはMath.floorなどを使って整数値にする必要があります。

ver.5ブラウザでも動くクロスブラウザDHTML

高橋登史朗　Takahashi Toshiaki

Webページを記述する言語HTML。DynamicHTMLの登場によって動的なページも多くなってきた。しかし，ブラウザの仕様の違いによる表示の食い違いも多く存在する。そこで，できるだけ多くのブラウザで互換性を持った用法を追究したのがクロスブラウザDHTMLだ。最新のブラウザに対応した互換性の高いページ記述法を紹介しよう。

DHTMLはNetscape 4.x（以後NN4）とInternet Explorer 4.x（IE4）から搭載されたブラウザベースの技術で，Dynamic HTMLの略である。DHTMLを使うと，Webページに貼り込まれた画像や文字やボタンやフォームなどのHTML要素をレイヤー化して見えなくしたり，見えるようにしたり，位置を動かしたり，重ね合わせたりということが自由に，好きなタイミングで，しかも，ピクセル単位の絶対位置指定に従って厳密にデザインできるようになる。それでいてプラグインもヘルパーアプリも必要ない。HTMLを書くのと同様にテキストエディタがあれば作れて，デフォルトのブラウザだけで即座に見ることができる。

つまり，DHTMLさえ使えれば，ちょっとしたWebアプリならサクサクッと作れてしまうのだ。ん〜，これは便利。でも，実は，

そんなに簡単じゃないのよ，これが。

なにしろDHTMLという確たる言語が存在していたわけではなく，なんとなくダイナミックかなぁという技術群をNetscapeとマイクロソフトがそれぞれ勝手に定義していたりする部分もあったりするものだから，HTMLやJavaScriptに毛が生えた程度のつもりで気軽にかじろうとすると，いくつものお約束の落とし穴にはまることになる。

最大の問題は，2大ブラウザのDHTMLについての解釈と実装が若干違っているために，この2大ブラウザ互換で使えるDHTMLを記述するには，メーカー側のリファレンスにはないテクニックを独自に編み出す必要があるというあたりだ。でも，

なけりゃ，作るさとゆ〜ことで……

ユーザーが自前でこしらえた互換テクニックを使ったページが徐々に普及してきている。そして，この2大ブラウザを互換させるテクニックは，クロスブラウザDHTMLと呼ばれているが，このようにして書くことによって初めて，1本のコードで，MacでもWinでもUNIXでもNetscapeでもIEでも動くダイナミックコンテンツを作ることができるのだ。やっぱX（クロス）でなくっちゃね，INTERNETなんだから。Oh!Xだし（違うか）。

さて，まずDHTMLを学ぶにはHTMLのほかにCSSとScript（主にJavaScript）が使えなくてはならない。まず，そこから入ってみよう。もちろん，ここで扱うのはクロスブラウザDHTMLだ。

DHTMLの基本構成

DHTMLの定義はブラウザメーカーによって若干異なるのだが，基本的な構成要素を多少強引に取り出すとおおむね次の3つになる。逆にいうとクロスなDHTMLを書きたければ，ここがポイントということになる。ざっくりいうと……。

①CSSでレイアウト定義された
②HTMLのタグ（オブジェクト）を
③JavaScriptなどを使ってダイナミックにコントロールする
　（消したり動かしたり色を変えたり）

という仕組みである。

次のサンプル（CD-ROM sample1）は「うごけっ」の部分のリンクをクリックするとリンク自体がブラウザ画面の左端から150ピクセルの位置へ移動するものだ。SCRIPTの中のleft = 150の部分がその動きを指示している。left = 150という行が3つもあるのはver.5のブラウザ（IE5/Gecko）とNN4.xとIE4.xそれぞれのDOMの構造が違うために指定方法がそれぞれ異なるためだ。将来的にはW3C-DOMに準拠しているver.5ブラウザの書き方が標準になる可能性が高いのだが，まだ少なくともしばらくはver.4ブラウザとクロスさせた書き方が必須となるだろう。

```
(CD-ROM sample1) HTML
<HTML>
<HEAD>

SCRIPT  <SCRIPT>
  function ugoke () {

    //ver.5 (IE5,Gecko) 用
    if (document.getElementById)
      document.getElementById ('sample1') .style.left=150

    //NN4用
    else if (document.layers)
      document.layers['sample1'].left=150

    //IE4用
    else if (document.all)
      document.all ('sample1') .style.left=150

  }
</SCRIPT>

CSS  <STYLE>
  #sample1 { position:absolute }
</STYLE>

HTML  </HEAD>
<BODY>

制御されるHTML  <DIV ID="sample1">
  sample1
<A HREF="#" onClick="ugoke () ">うごけっ</A>
</DIV>

HTML  </BODY>
</HTML>
```

CSSの定義方法

CSSはHTMLで定義された文章構造にレイアウトなどの定義をするものだがその定義の方法には次の3つがある。例はどれもTESTという文字の

フォントサイズを200pxに指定する同じ定義のバリエーションで，ブラウザで見るとどれも図1のように大きな文字が表示される（CD-ROM sample2）。

1.タグ内のSTYLE属性で定義する

```
<DIV STYLE="font-size:200px">TEST</DIV>
```

2.STYLEタグで定義する

```
<STYLE TYPE="text/css">
  #test2 { font-size:200px }
</STYLE>

<DIV ID="test2">TEST</DIV>
```

3.外部ファイルから読み込んで定義する

```
<LINK REL="stylesheet" HREF="style.css">
<DIV ID="test3">TEST</DIV>

外部ファイルstyle.cssの中身
#test3 { font-size:200px }
```

図1

上記1～3の方法による定義の適用優先順位はHTML<3<2<1の順となる。たとえば，1のタグの中に直接書き込む方法は，使いすぎると見にくくなるが，優先度がもっとも高いので，細かい仕上げ処理などには向いている。またCSSが指定されるとHTMLによる同種の定義（たとえばCSSのfont-sizeに対するHTMLのFONT SIZE）は無視される。

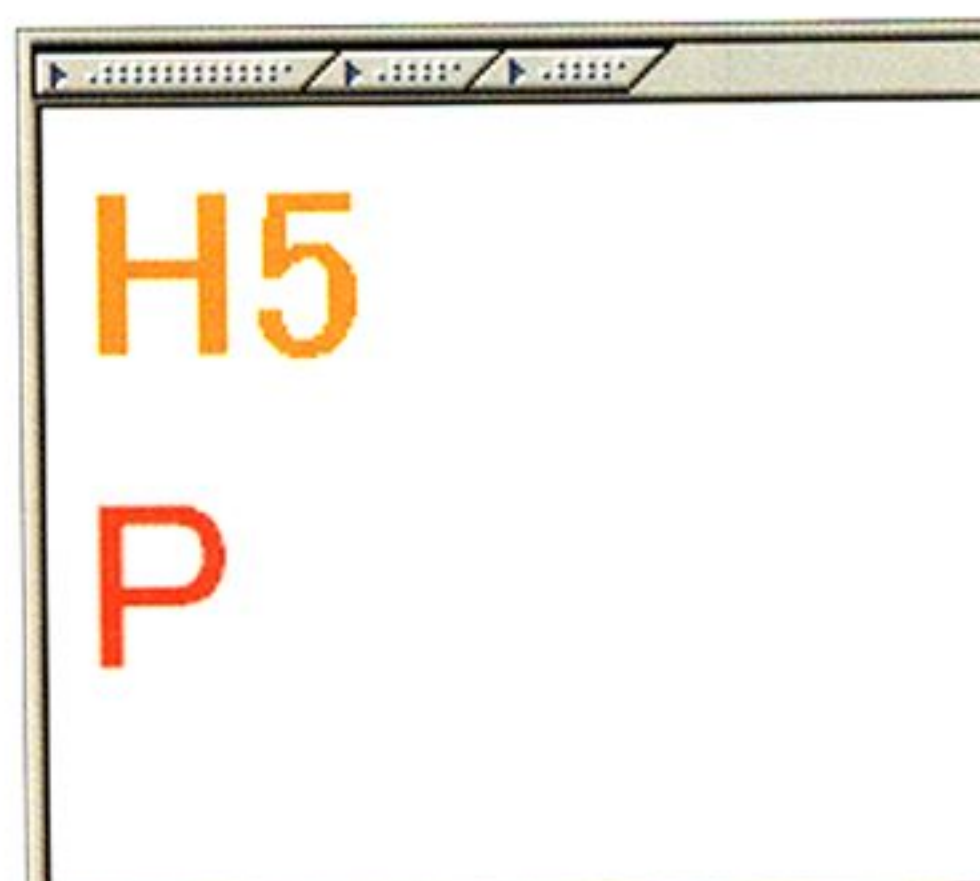
セレクタ

前出の例の中の#testという文字はセレクタと呼ばれる部分で，#で始まるセレクタはHTMLタグの中のID名に対応している。これによって，testというIDのついたタグのCSS属性を定義するわけだ。セレクタにも種類があるが，DHTMLで使うときは個々のタグをコントロールするためにIDセレクタが使われる場合が多い。主なセレクタは次のとおり（CD-ROM sample3）。

1.タグを使うセレクタ（図2）

タグにスタイルを指定する。指定されたHTML文章中のタグすべてに適用される。例ではBODYタグ内のフォントサイズをすべて50pxにして，PとH5タグの文字の色をそれぞれ指定している。

```
<HTML>
<STYLE TYPE="text/css">
BODY    { font-size:50px  }
H5      { color:orange    }
P       { color:red       }
</STYLE>
<BODY>
<H5>H5</H5>
<P>P</P>
</BODY>
</HTML>
```

図2

2.クラス属性を使うセレクタ（図3）

.（ドット）で始まるセレクタと同じCLASS名を持つタグのスタイルを定義する。これによって，任意の複数のタグをまとめて定義することができる。例は，abcという名前のついたタグのフォントサイズを20pxに指定しているが，同時にDIVタグで.abcというクラス名のものだけ別に100pxになるようにしている。

```
<HTML>
<STYLE TYPE="text/css">
 .abc      { font-size:20px  }
 DIV.abc  { font-size:100px }
</STYLE>
<BODY>
<SPAN CLASS=abc>SPAN</SPAN>
<DIV CLASS=abc>DIV</DIV>
<P CLASS=abc>P</P>
</BODY>
</HTML>
```

図3

3.ID属性を使うセレクタ（図4）

#で始まるセレクタとID名を持つ個々のタグのスタイルを定義する。例は，abcというID名のついたタグのフォントサイズを20pxに指定し，defというID名のものを100pxにしている。複数のタグを一度に指定したいときはIDではなくCLASSを使う。

```
<STYLE TYPE="text/css">
  #abc   { font-size:20px  }
  #def   { font-size:100px }
</STYLE>
<BODY>
<SPAN ID=abc>#abc</SPAN>
<SPAN ID=def>#def</SPAN>
</BODY>
</HTML>
```

#abc

#def

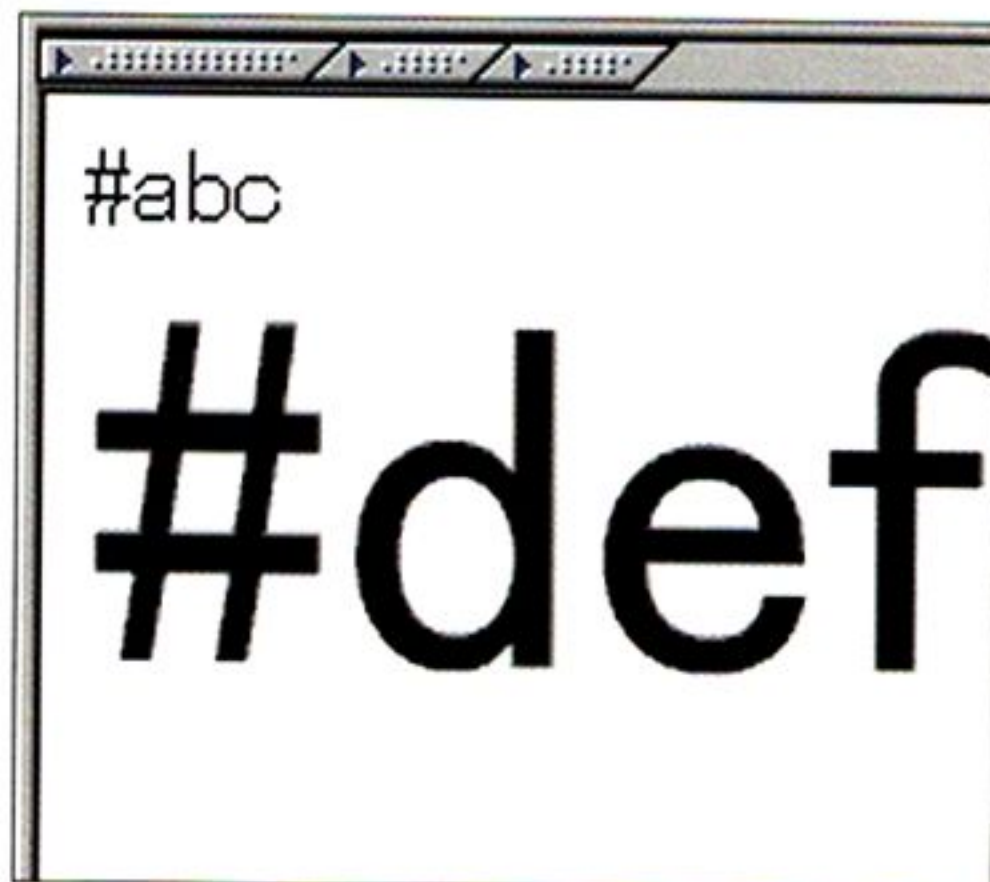



図4

CSS-Pを使ってみよう

さて，ここまではCSS全般の説明だったが，ここからはCSSの中でDHTMLにとってもっとも大事な部分，CSS-Pに絞り込んで見てみよう。CSS-PとはW3C（World Wide Web Consortium）によって1997年1月と8月に草案化されたPositioning HTML Elements width CSSのことだ。CSS-Pはver.4ブラウザ（NN4，IE4）のDHTMLの基本にかかわる部分で，それはver.5ブラウザ（IE5，Gecko）の中にも吸収されているから，ここを覚えておけば，ver.4ブラウザ同士をクロスで使うことはもとより，ver.4とver.5ブラウザをも共通で動かすための役に立つだろう（※CSS-Pの内容はその後，1998年に勧告されたCSS2へ織り込まれている）。

1.位置を動かしてみる（CD-ROM sample4）

CSSはtopとleftのプロパティで位置を指定できる。さらにそれをJava Scriptで自由に動かしてみよう。次のサンプルはsample1をmoveLAYER（layName，x，y）という関数にまとめて汎用化したものだ。

まず，

```
STYLE="position:absolute;top:10;left:10"
```

と書くことでID名"sample4"のタグをレイヤー化し画面上端から10px，左端から10pxの位置に表示している（図5）。

そこで，この「うごけっ」の文字をクリックすると，

```
moveLAYER ('sample4',35,100)
```

が起動してレイヤー'sample4'を画面左端から35px，上端から100pxの位置へ移動させることができる（図6）（HREF="#"は便宜上リンクを無効化しただけ）。

スクリプトの中の，if（document.getElementById）はver.5ブラウザ（IE5/Gecko）用の処理で，if（document.layers）はNN4用，if（document.all）はIE4用の処理だ。これで，NN4でもNN5でもIE4でもIE5と同じように動くクロスなDHTMLのできあがりということだ。

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE='JavaScript'>
<!--

function moveLAYER (layName,x,y) {

  //--For ver.5 Browsers
  if (document.getElementById) {
    document.getElementById (layName) .style.left=x
    document.getElementById (layName) .style.top=y
  }

  //--For ver.4 Browsers
  else if (document.layers) document.layers[layName].
                                          moveTo (x,y)
  else if (document.all) {
    document.all (layName) .style.pixelLeft=x
    document.all (layName) .style.pixelTop=y
  }
}

//-->
</SCRIPT>
</HEAD>
<BODY>

<DIV ID="sample4" STYLE="position:absolute;top:10;left:10">
 sample4
<A HREF="#" onClick=
         "moveLAYER ('sample4',35,100) ">うごけっ</A>
</DIV>

</BODY>
</HTML>
```

sample4 うごけっ

図5

sample4 うごけっ

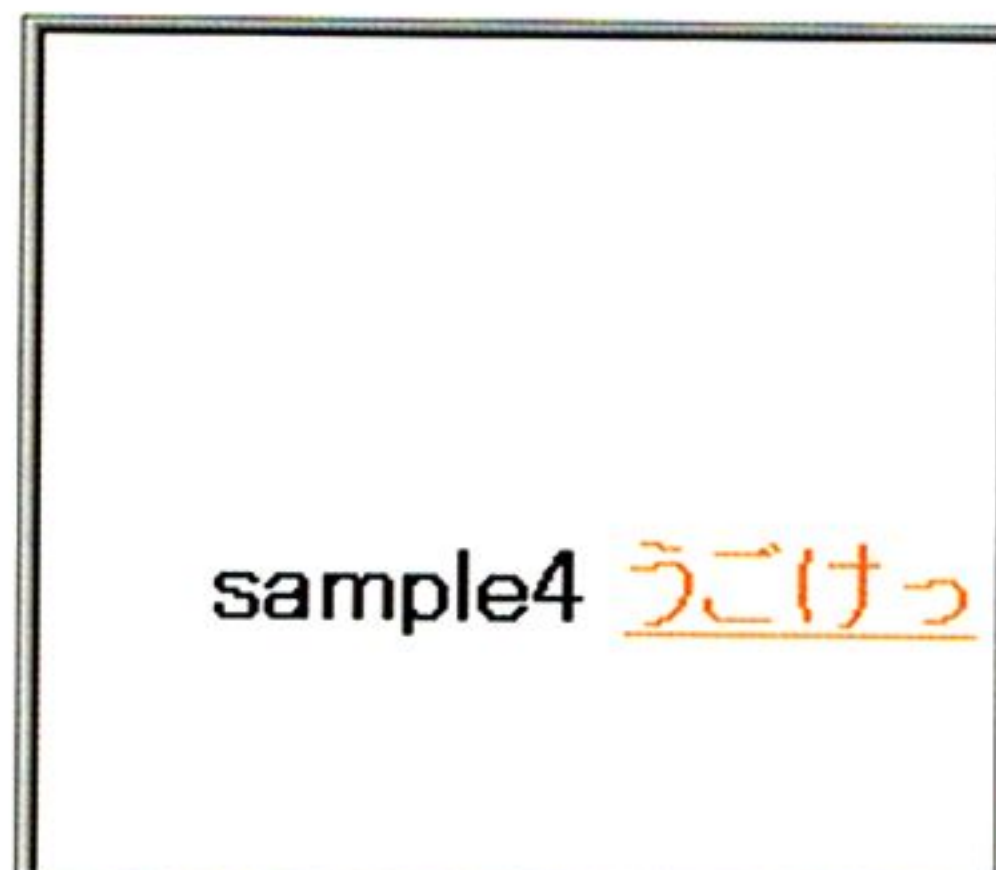



図6

2.消したり，出したりしてみる（CD-ROM sample5）

visibilityを使うと表示属性を制御できる。つまり，画面上から消したり，出したりしてみせられるというわけだ。もちろん，前述のtop/leftを-1000pxなど画面の外に設定すれば見えなくなるが，visibilityはそのレイヤーブロックを同じ場所に置いたままで隠すことができる。

次のサンプルは，この表示/非表示処理をvisibleLAYER（layName，switch）という関数にまとめて汎用化している。まず，位置を動かすsample4と同様に，

```
STYLE="position:absolute;top:10;left:10"
```

と書くことでID名"sample5"のタグをレイヤー化してスタンバイしている（図7）。あとは「消えろっ」をクリックすると，

```
visibleLAYER ('sample5',0)
```

が起動してレイヤー'sample5'を画面上から見えなくしてくれる（図8）。

次に「現れろっ」をクリックすると今度は，

```
visibleLAYER ('sample5',1)
```

が起動してレイヤー'sample5'が画面上に現れる（図7）。引数の0で非表示，それ以外で表示の属性を設定できるようになっている（HREF="#"は便宜上リンクを無効化しただけ）。

スクリプトの中のif（document.getElementById）はsample4と同様に

ver.5ブラウザ(IE5/Gecko)用の処理でif(document.layers)はNN4用，if(document.all)はIE4用の処理だ。

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE='JavaScript'>
<!--

function visibleLAYER (layName,swt) {

  if (swt==0) var visibleSwt = 'hidden'  //--非表示
   else       var visibleSwt = 'visible'  //--表示

  //--For ver.5 Browsers
  if (document.getElementById)
   document.getElementById (layName) .
                              style.visibility=visibleSwt

  //--For ver.4 Browsers
  else if (document.layers)
   document.layers[layName].visibility=visibleSwt
  else if (document.all)
   document.all (layName) .style.visibility=visibleSwt

}

//-->
</SCRIPT>
</HEAD>
<BODY>

<DIV ID="sample5" STYLE="position:absolute;top:10;left:10">
 sample5
</DIV>

<DIV ID="hide" STYLE="position:absolute;top:80;left:30">
 hidden
<A HREF="#" onClick="visibleLAYER ('sample5',0) ">
                                             消えろっ</A>
</DIV>

<DIV ID="show" STYLE="position:absolute;top:100;left:30">
 visible
```

表　CSS-P（抜粋）

プロパティ		値	使用例	初期値
position	HTMLエレメント（タグ）の表示位置の決め方を指定する	static(通常,) ¦ relative(相対位置指定) ¦ absolute(絶対位置指定) ¦ fixed(固定/CSS2) ¦ inherit (親タグから継承)	<STYLE> #t { position:absolute } </STYLE> <DIV ID="t"></DIV>	static
left	画面または親タグの左端からタグの左端までの距離	<数値+単位> ¦ <%> ¦ auto ¦ inherit	<STYLE> #t { position:absolute; left:100px } </STYLE> <DIV ID="t"></DIV>	auto
top	画面または親タグの上端からタグの上端までの距離	<数値+単位> ¦ <%> ¦ auto ¦ inherit	<STYLE> #t { position:absolute; top:50pt } </STYLE> <DIV ID="t"></DIV>	auto
width	HTMLエレメントの幅	<数値+単位> ¦ <%> ¦ auto ¦ inherit	<STYLE> #t { position:absolute; width:500 } </STYLE> <DIV ID="t"></DIV>	auto
height	HTMLエレメントの高さ	<数値+単位> ¦ <%> ¦ auto ¦ inherit	<STYLE> #t { position:absolute; height:200px } </STYLE> <DIV ID="t"></DIV>	auto
clip	HTMLエレメントの一部を範囲指定して表示する	<shape> ¦ auto ¦ inherit	<STYLE> #t { position:absolute; width:100px;height:100px; clip:rct (10px,20px,30,10px) } </STYLE> <DIV ID="t"></DIV>	auto
visibility	表示/非表示を指定する	visible(表示) ¦ inherit ¦ hidden(非表示) ¦ collapse(CSS2)	<STYLE> #t { position:absolute; visibility:hidden } </STYLE> <DIV ID="t"></DIV>	inherit
z-index	HTMLエレメントどうしの重なりを0から始まる整数で指定する(大きいものが上)	auto ¦ <整数> ¦ inherit	<STYLE> #t { position:absolute; z-index:5 } </STYLE> <DIV ID="t"></DIV>	auto

```
<A HREF="#" onClick="visibleLAYER ('sample5',1) ">
                                        現れろっ</A>
</DIV>
</BODY>
</HTML>
```

sample5

hidden 消えろっ
visible 現れろっ

図7

hidden 消えろっ
visible 現れろっ

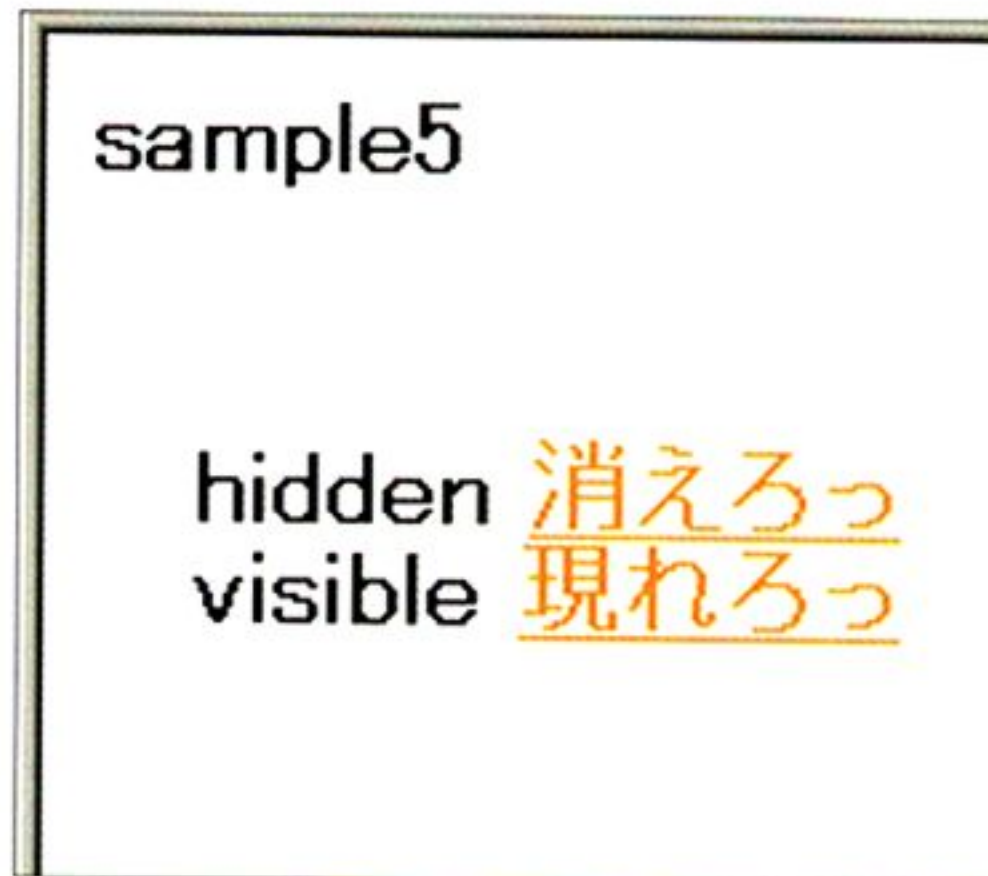



図8

応用サンプル/ポップアップヘルプ

DHTML関連の命令群を紹介しはじめるとOh!X 1冊まるまる使っても足りないくらいなので，とりあえず，今回の締めとして1本，応用サンプルを取り上げておく。もっと詳しく知りたい人は，後出のURLや書籍にあたってみるのがいいだろう。

次のサンプルは，リンクに触ると影付きのヘルプレイヤーが現れるというものだ。各リンクの跳び先や処理内容の説明メモなどを書き込んでおくと重宝するスクリプトだ。

ポップアップヘルプ(CD-ROM sample6)

このスクリプトは2つのファイルに分かれている。ひとつは，HTMLで，触るとヘルプを表示するリンクを書き込む本体部分だ。このなかに，もうひとつの.js外部ファイルにまとめたヘルプスクリプトを読み込んで使っている。外部ファイルの内容を直接HTMLの<SCRIPT>タグの中に貼り込んでも使えるわけだが，スクリプトの性質上，いろいろなページで使い回すことが多いので，共通する処理を外部ファイルへまとめて分けてある。これで，メンテナンスが格段に向上するわけだ。

まず，htmlの中はリンクとヘルプレイヤーに分かれている。ヘルプレイヤーは影用のレイヤーとヘルプ用のレイヤーを用意し，SCRIPT SRCでhelp.js外部ファイルを読み込んでいる。リンクのほうでは，onMouseOver(マウスポインタがリンクに触ること)でヘルプメッセージを変数msgへセットしてから，

```
showHELP (msg,event)
```

が起動するようにしている(図9)。if(document.layers||document.all)で分岐しているのはdocument.layersの使えるNN4とdocument.allの使えるIE4,5以外のブラウザでeventの文字に反応したエラーが出ないようにするためだ。次に，onMouseOut(マウスポインタがリンクから離れること)で，

```
hideHELP ()
```

が起動して，ヘルプを消すようになっている。

help.jsのなかは，クロスブラウザのための関数群とヘルプを表示するshowHELP(msg,event)，ヘルプを消すhideHELP()の3つのブロックで構成されている。将来のブラウザバージョンアップやバグ対応は関数群を修正することで対応することで修正処理を簡素化する仕組みだ(ただ，残念ながらここで掲載するバージョンはまだGeckoには対応していない。IE5とIE4とNN4は動作するので，少なくとも現時点ではそのまま使ってほとんど問題はないが，バージョンアップは筆者のサポートページで行っているのでそれを参照のこと)。

```
html
<HTML>
<HEAD></HEAD>
<BODY>

<A HREF="samples/01/sample6.htm"

 onMouseOver="
   msg='<B>Sample6 : </B>マウスポインタが
                    リンクに触ると，影付きヘルプが現れる。';
   if (document.layers||document.all)
                                 showHELP (msg,event) ;
   return true"

 onMouseOut ="hideHELP () ;return true"

> Sample6 </A>

<!--HELPレイヤー-->
<DIV ID="helplaysdw" STYLE="position:absolute"></DIV>
<DIV ID="helplay" STYLE="position:absolute"></DIV>
<SCRIPT LANGUAGE="JavaScript"
                        SRC="tools/helplay.js"></SCRIPT>

</BODY>
</HTML>
```

```
help.js
/* -----------------------------------------------------------------
 * Cross-Browser Functions クロスブラウザのための関数
 * -----------------------------------------------------------------
 * サポート http://www.fureai.or.jp/~tato/JS/BOOK/INDEX.HTM
 */

var nn4=document.layers  //Netscape4.x
var ie4=document.all     //IE4.x


 //--マウスポインタのX座標取得
 function getMouseX (e) {
   if      (nn4) return e.pageX
   else if (ie4) return document.body.scrollLeft+event.clientX
 }

 //--マウスポインタのY座標取得
 function getMouseY (e) {
   if      (nn4) return e.pageY
   else if (ie4) return document.body.scrollTop+event.clientY
 }
```

```
//--レイヤーを動かす
function moveLAYER (layName,x,y) {
  if      (nn4) document.layers[layName].moveTo (x,y)
  else if (ie4) {
        document.all (layName) .style.pixelLeft=x
        document.all (layName) .style.pixelTop=y
  }
}

//--背景色セット
function setBGCOLOR (layName,color) {
  if      (nn4) document.layers[layName].bgColor=color
  else if (ie4)  document.all (layName) .style.
                                     backgroundColor=color
}

//--HTML出力
function outputLAYER (layName,html) {
  if      (nn4) {
     with (document.layers[layName].document) {
       open ()
       write (html)
       close ()
     }
  }
  else if (ie4) {
    document.all (layName) .innerHTML=html
  }
}


/*
 * ヘルプを出す
 */

 function showHELP (msg,e,swt) {

 //--HELP TABEL
 var msgtbl ='<META HTTP-EQUIV="Content-Type">'
     msgtbl+ ='     CONTENT="text/html; charset=x-sjis">'
     msgtbl+ ='<TABLE BORDER=1 WIDTH=248><TR><TD>'
     msgtbl+ ='<TABLE BORDER=1 BGCOLOR=#cccc99
                                   WIDTH=240><TR><TD>'
     msgtbl+ ='<FONT COLOR="#222222">'
     msgtbl+ ='<IMG SRC="tools/help.gif" BORDER=0
                                   WIDTH=18 HEIGHT=16>'
     msgtbl+ =msg
     msgtbl+ ='</FONT>'
     msgtbl+='</TD></TR></TABLE></TD></TR></TABLE>'
 outputLAYER ('helplay',msgtbl)

 var msgtbl ='<META HTTP-EQUIV="Content-Type">'
     msgtbl+ ='     CONTENT="text/html; charset=x-sjis">'
     msgtbl+ ='<TABLE BGCOLOR=#222222
                         BORDER=0 WIDTH=242><TR><TD>'
     msgtbl+ ='<TABLE BGCOLOR=#222222
                         BORDER=0 WIDTH=234><TR><TD>'
     msgtbl+='<FONT COLOR="#222222">'
     msgtbl+=msg
     msgtbl+='</FONT>'
     msgtbl+='</TD></TR></TABLE></TD></TR></TABLE>'
 setBGCOLOR ('helplaysdw','#000000')
 outputLAYER ('helplaysdw',msgtbl)

   var offsetx=10      //ヘルプ位置をマウスポインタから
                                   左右へ何ピクセル離すか
   var offsety=-100  //ヘルプ位置をマウスポインタから
                                   上下へ何ピクセル離すか
   var offsetxsdw=20 //影をヘルプから左右へ何ピクセル離すか
   var offsetysdw=20 //影をヘルプから上下へ何ピクセル離すか
   moveLAYER ('helplaysdw', (getMouseX (e) +offsetx+
          offsetxsdw) , (getMouseY (e) +offsety+offsetysdw))
   moveLAYER ('helplay', (getMouseX (e) +offsetx) ,
                                   (getMouseY (e) +offsety))

 return true
 }

/*
 * ヘルプを消す
 */

 function hideHELP (e) {
   moveLAYER ('helplaysdw',-200,-200)
   moveLAYER ('helplay',-200,-200)
   outputLAYER ('helplay','')
   return true
 }
```

Sample6

図9

DHTML関連サイト

［DHTML］

Netscape/Dynamic HTML Developer Central
http://developer.netscape.com/tech/dynhtml/index.html

Microsoft/DHTML, HTML & CSS
http://msdn.microsoft.com/workshop/c-frame.htm#/workshop/author/default.asp

［CSS2］

Cascading Style Sheets, level 2 CSS2 Specification（W3C REC-CSS2-19980512）
http://www.w3.org/TR/REC-CSS2/

［CSS-P］

Positioning HTML Elements with Cascading Style Sheets（W3C WD-positioning-19970819）
http://www.w3.org/TR/WD-positioning/

［DOM］

Document Object Model（DOM）Level 1 Specification（W3C REC-DOM-Level-1-19981001）
http://www.w3.org/TR/REC-DOM-Level-1/

［日本］

古籏一浩氏のページ（リファレンスが充実）
http://www.shiojiri.ne.jp/~openspc/JavaScript/index.html

萩原氏のページ（Tipsが充実）
http://www.din.or.jp/~hagi3/JavaScript/JSTips/Default.htm

森山和広氏のページ（ゲームが充実）
http://plaza.harmonix.ne.jp/~jimmeans/

半場方人氏のページ（リンクが充実）
http://www.bekkoame.or.jp/~hamba/link/jslink.html

NewGameWeb（ブラウザゲームの殿堂）
http://web01.fureai.or.jp/~tato/GameWeb/index.cgi

高橋登史朗（筆者）のJavaScript/DHTMLサポートページ
http://www.fureai.or.jp/~tato/JS/BOOK/INDEX.HTM
http://www.fureai.or.jp/~tato/DHTML/simple/contents.htm

オルタナ・ピコピコC言語講座＃0
とりあえずプログラムを組んでみよう

飯田 伸一　Iida Shinichi

Windows上のゲームプログラミングといっても，構える必要はない。elライブラリを使用すれば面倒な部分をすべてユーザーの作業から隠してくれる。Visual C++の勉強の一環として，ここでは贅沢にDirectXを使って，ピコピコゲームを作ってみよう。

まずは脱線

最近の学校教育では「ガマン」という言葉は流行らないらしい。なにをやるにしても，楽しさがなければ，生徒はついてこないというのだ。なんだか楽しいことはすべてよいことだと勘違いしているのは子供たちだけではなく，教える立場にいる教育関係者にも多いようだ。「楽しさが知識になる」というひと昔前のパソコンCMのコピーが，いまだに通用しそうで怖い[*1]。

かつての管理教育の反動でこうなってしまったらしいが，どうも日本人というのは，極端に走るまでコトの重要性に気がつかない傾向にあるようだ。

あの管理教育で教えていたのは，システムに隷属するための極端に後ろ向きな「ガマン」であり，その教育手段として，教師による体罰が正当化され，学校と刑務所との違いがわからなくなってしまうほどヒドイ結果となった。その反動からだろうか，今度は逆に「個性」を重視しようということで，生徒にはまだ理解できていない無制限な「自由」が与えられてしまった。

その結果，現在の小学生を15分だけでもじっとさせることすら，困難になるという異常な事態を招いてしまったのだ。

はっきりいって，よい意味での「ガマン」は必要だ。なにか明確な目標があって，それに行きつくまでの過程には，前向きに耐えなければならない時期が必ずある。成功というのは，自己の失敗を乗り越え，耐えぬいたものに与えられる称号なのだ(奇跡なんてのは信じてはいけない，所詮人生なんて連続的なものなのだから……)。

＊1　「感動が知識になる」だっけ？　10年以上も前のことだから，忘れてしまったよ。ま，時効ということで……。

Hello, hello, how low world?

前文と矛盾するようだが，この講座で取り上げるC言語の学習方法は，はっきりいって「ガマン」を必要としない(僕の文章を読むほうが忍耐を必要とするかも……って自爆)。

簡単なWindows上で走るピコピコゲーム制作を通して，実践的にC(とC++を少し)を学んでもらうということを主眼においている。それは，C学習の入り口としていちばんわかりやすいからだ。"Hello, world!" で始めるよりは，よっぽど面白いだろう。[*2]

伝統的なCの解説書とはまったく異なるスタンスをとるという意味で，当講座のタイトルに「オルタナ」(伝統的な手法とはまったく違うという意味)という言葉を入れたのだ。具体的には，僕が作ったゲームをサンプルにCを解説していくというかたちで話を進めるわけだが，もちろん各自Cの習熟度には差があると思う。すでに理解している項目については，ガンガン読み飛ばしてほしい。この講座でいったんCを学習し終わり，要領がわかればそれで全部OKということには当然ならない。それからは各自の事情により，いろいろな「ガマン」が要求されるはずだ(もちろん，楽しさも伴うものだが)。

＊2　人間というのは，なにかを学習したりするときには，必要性を感じなければ，飽きて途中で止めたり，仮に続けることができたとしても，たいして実用的な知識は身につかないものだ。趣味でCを学習したい場合は，"Hello, world!" の表示から始まって順番に文法解説を読まされるより，簡単なピコピコゲームを作りながら学んだほうがよいのに決まっている(←偏見＆暴言)。

ゲーム作りはピコピコゲームから

現在のゲーム制作の敷居は，技術・開発環境から見るととても高いものとなっている。初心者がいきなり，最新のポリゴン技術などを駆使したゲームなんて作れるはずがないし，「さーて，ゲームでも作ってみるか」と，意気揚々と大枚はたいてコンパイラを買ったはいいが[*3]，どうしたらゲームを作れるかといった基本的なことすらわからなくて，途方に暮れてしまっている人が多いというのが現状なのだ。

その点，ピコピコゲーム(要するにピコピコという効果音がもっとも似合うようなゲーム)は，一見地味に見えるかもしれないがゲームを作るための必要な手法を学ぶ格好の教材となりうるだろうし，以下に述べるelライブラリを使用することによって，さほど開発環境の知識がなくてもゲームプログラミングを行うことが可能になるのだ。

自分でピコピコゲームのプログラムを，あれこれといじくったり，実際に自分でゲームを作ることによって，少しずつCとゲームプログラミングのコツをつかんでいけることだろう。あとは各自必要に応じて細かいことを覚えていけばいい。

C言語の講座を始めるにあたって，1冊はいつでも必要なときに標準関数などを参照できる辞書的な参考書の購入をオススメする。疑問があったら，そちらの詳しい解説を見てほしい。なぜなら当講座ではゲームを作成しながらCの勉強をするという乱暴(無謀ともいう)な手法を用いているので，最低限のことしか説明できないからだ。

＊3　僕が使用しているのは，マイクロソフトのVisual C++ 5.0のLearning Editionと6.0Professional Editionである。両方とも僕が大学生のときに買ったアカデミックパックだが，バージョンが6.0になってからアカデミックパックでも商利用が可能になった(電話で直接マイクロソフト社に確かめた)。一般版とは，ビビリが入るほどの価格差があるので，学割がきく，学生のうちにゲットするのが正解だ(マジ安いっス……)。

elを用いてDirectXを簡単に利用する

Windows上でゲームを作成するにはDirectX[*4]を使用するのが一般的だ。このDirectXの仕組みは複雑であり，覚えなくてはならないことがあまりにも多いので初心者にとっては大きな壁となっている。これではCの学習とゲーム制作という本来の目標にたどり着く前に相当な労力と時間が必要となってしまう。Cで気軽にゲームを作ろうとしても，いくらなんでもこれでは，無理があるというものだ。

そこで，当講座ではelというDirectXをシンプルに利用できるライブラリを用い，Cの学習を進めることにする。それがelライブラリだ(コラム1を参照)。このelを利用することによって，Cの知識だけで，比較的簡単にゲームが作れるようになる。

これは，あくまでも僕の意見だが，本格的にゲームプログラマを目指す人でも最初はelを通してCとゲームプログラミングの基礎を学び，それからDirectXを直接使う方向へステップアップしたほうが，効率的だと考えている。さらに付け加えると，elはDirectXを使わない部分でもかなり高度なことができるので，特に趣味でゲームを作るユーザーの場合には(いわゆ

る暇プロ……懐かしいフレーズだ)，elだけで十分だと思う。

elの使い方は，順次Cの解説とともにしていくが，当講座の主目的はCを学習することなので，elについては断片的にしか説明していない。そこで，詳細については，elヘルプを参照してもらうことになるのだが，本文の解説だけでも十分に理解できるように配慮しつつ解説していくつもりだ。

※elとelヘルプはCD－ROMに収録
＊4　DirectXとは，ゲーム中での映像の描画，キーボード/マウスなどの入出力，音楽の演奏などを行うための機能だと，単純に考えてもらえば間違いないだろう。

さてさて，本題へ……

まずは，リスト1を見てもらおう。ゲームのメイン処理関数MainScreenから，正味たった70行程度のコードで構成されているが，これだけで一応ゲームとしての体裁を保っている。Windowsにおけるゲームプログラムとしては，驚異的な短さだ(実行ファイルが100Kバイトを超えているところに，Windowsの業の深さを感じてしまうのだが……)。

これは，僕が試しに初めてWindowsで作ったゲームなのだが，あまりにもあっさりと完成してしまったので，なんだか拍子抜けしてしまったほどだ。ま，こんなに簡単にできてしまったのは，ひとえにelのおかげだが，とにかく細かいことにこだわらないでCの命令セットとゲーム作りの方法が同時に学べるという，都合のよい方法が見つかったというわけだ。

このゲームは"Snaky1"という名前の単純なヘビ(ボ？)ゲームだ(画面1)。どんどんヘビの体が長くなっていくので，自分の体と壁に当たらないようにカーソルキーで操作してほしい。なお，エスケープキーでゲーム終了だ。

プログラム的にはそんなに難しいことはしていない。ただ，当たり判定のところで，昔懐かしい「仮想画面」というのを使っているくらいだ。これはあとで詳しく説明する。

では，リスト1の解説をプログラムの流れ(表1)に沿って始めることにしよう。本文の説明とリストを読みながら，学習していこう。

●コメントはできるだけ多くつけよう

まず，最初の数行はコメントという部分で，あとでプログラムを見たときにわかりやすいように，説明などを書いておくためのものだ。コンパイル時には無視される部分だ。「/＊」と「＊/」で括られた範囲がコメントとなる。

例：/＊コメント ＊/

プログラム先頭のコメントは，通常「プログラムの名前」，「制作日時」，「署名」などを入れる。たとえ自分ひとりしか，そのプログラムを見ることがなくとも，このコメントは必ず入れておいたほうがいい。プログラムをたくさん組むにつれ，「こんなの作ったっけ？」と忘れることがあるからだ(←経験者)[＊5]。

なお，「//」だけで，そこから行末(改行)までをコメントとすることもできる。これは特に簡単なコメントをつけたり，一時的にその行だけを無効にしたいとき(デバッグ時とか)に便利だ。

＊5　僕は，ファXXキン・グレイトな忘れっぽい頭脳の持ち主で，大学時代に女の子の友達に「いつご飯たべたっけ？」と聞いたら「さっき……」と答えられたことがある。この一件で僕の評判はもう「うなぎ下がり(？)」になってしまった。

el(Easy Link Library)について

COLUMN

elとは，Botchy氏が開発したフリーウェアで，DirectX(5.0以上推奨)を使用して簡単にゲームを作れるという非常に優れたライブラリだ。難しくて面倒なDirectXとWindowsの処理を隠しているので，簡単なC言語の知識だけで，手軽にゲームを作ることができる。

対応しているコンパイラは，マイクロソフトのVisual C++(ver4.0以上)。インプライズのBorland C++，C++Builder(ただし一部に機能制限あり)である。

elを使用するためには，それぞれの開発環境によって設定方法が異なる。その方法は，el開発者による親切なヘルプに(¥El¥elhelp.hlp)にわかりやすく書いてあるので，各自の開発環境に応じた設定をしてもらいたい。

Oh!Xでのelを使用したC言語講座とライブラリの再頒布を快く承諾してくださったBotchy氏の寛容さには，ひたすら頭が下る思いだ。非常に高度なことにも対応している有用なライブラリであるのにも関わらず，elはすべてにおいてフリーというBotchy氏の姿勢に，感謝の意を表したい。

※Botchy氏のホームページ，http://www.bekkoame.ne.jp/ha/botchy/

●プリプロセッサ制御文・#include, #define

#で始まるプリプロセッサ制御文は，コンパイラがプログラムをコンパイルする前に処理される命令である。指定したファイルを取り込んだり，文字列の置き換えを行うことができる。

・#include"el.h"

el.hのファイルがコンパイル後，プログラムに挿入される。

ここで取り込まれるファイルは，ヘッダファイルと呼ばれるものだ。これらのヘッダファイルを読み込むことによって，Cでいろいろな関数が使えるようになる。つまり，C単体ではほとんどなにもできないのだ。

Cは関数の組み合わせによってプログラムが構成されている。早い話，計算式以外で()が含まれるものはほとんど関数だといえる。

ここでいう関数とは，画面に文字を表示するとか，キーボードから文字を入力するといった処理をする一連の命令のようなものだと考えてほしい。最初にいっておくと，頭にelのつくものは(elLoop,elDraw etc.)，elライブラリ(クラス)であり，通常のCの関数群と簡単に見分けることができる。

el.hのヘッダファイルの中身を見てみると，Cのシステム標準関数であるstdio.hがインクルードされているのがわかるだろう。stdio.hは，Cでプログラムを開発する際にもっとも使用頻度が高い関数が集められたものだ。

・#define MAX_X 38

#defineは，後ろに指定された数値，文字列の置き換えを行う。#define MAX_X 38となっているが，これ以後はMAX_Xと記述すれば，38という数値に置き換わる。たとえば，a＝MAX_X;とすると，変数aの値は38となる(変数の扱いについては後述)。

#defineを使用するメリットとしては，プログラム中で何度も出てくる定数や文字列の置き換えをしたい場合に，ただ1カ所#defineの部分を変更するだけで済むという点がある。

画面1

表1　メイン画面関数の処理の流れ

1　画面が切り替わった直後の処理…初期化関数InitFuncへ
(画面のクリア)
(各変数の初期化…スコア，ヘビの座標，仮想画面など)
↓
2　仮想画面の最新座標にヘビキャラクタ'@'を代入
↓
3　仮想画面の内容を(裏)画面に描画
↓
4　キー入力処理とヘビの向いている方向によってX,Y座標を増減
↓
5　ゲームオーバ？……壁や自分の体に当たって自爆した場合
(ゲームオーバ処理関数へ)
↓
6　スコアをひとつ足し，ハイスコア更新かをチェック
↓
7　裏画面を表画面に転送→再びelLoop経由でメイン関数の先頭へループ

リスト1

```
#include "el.h"

#define MAX_X 38            //  X座標の最大値
#define MAX_Y 28            //  Y座標の最大値

//  画面Noを定義
//  プログラムでは、直接画面の関数を呼び出すのではなく、この画面Noを使用する。
#define MAIN_SCREEN 1
#define OVER_SCREEN 2

//  上の画面Noと対になる、画面用の関数のプロトタイプ宣言
void MainScreen(void);
void OverScreen(void);

//  ゲーム処理中の初期化関数のプロトタイプ宣言
void InitFunc(void);

//  各変数の定義
int     Score=0;
int     Hiscore=100;
int i,j;
int D,Px,Py;                //  描画座標

char ImaginaryScreen[MAX_X][MAX_Y];

/*************************************************************************/
/*                                                                       */
/*  メイン関数(一般のCでは"main",Windowsでは"WinMain")                   */
/*                                                                       */
/*      プログラムが起動すると、最初に動き出す関数で、                   */
/*   DirectXの初期化等は、全てここで自動的に行われる。                   */
/*                                                                       */
/*************************************************************************/

//  パラメーターは1つで、プログラムの名前を入れる。
int elMain("Snaky1")
{
   //  以下の"elWindow"を削除するとフルスクリーン表示となる。
   elWindow(640,480,TRUE);

   elLoop()
   {
       //  先ほど定義した、画面No (MAIN_SCREEN) と、それと対になる画面用の関数
       // (MainScreen) を、定義する。後述の"elCallScreen"でMAIN_SCREENを指定すれば、
       // MainScreen関数が、繰り返し呼び出される (OVER_SCREENも同様)。
       elSetScreen(MAIN_SCREEN,MainScreen());
       elSetScreen(OVER_SCREEN,OverScreen());
   }

   //  elMain関数の終了
   elExitMain();
}

/*************************************************************************/
/*                                                                       */
/*  ウィンドウ生成関数                                                   */
/*                                                                       */
/*      ウィンドウが作られる時に、呼び出される関数で、解像度などを指定する。 */
/*                                                                       */
/*************************************************************************/

void elCreate(void)
{
   //  解像度を、640×480ドットに指定
   elDraw::Screen(640,480);

   //  最初に呼び出したい、画面Noを指定
   elCallScreen(MAIN_SCREEN);
}

/*************************************************************************/
/*                                                                       */
/*  キーボード関数                                                       */
/*                                                                       */
/*      何かキーが押されると、呼び出される関数。                         */
/*  この処理は、実際のゲーム画面とは別のタイミングで発生する。           */
/*  どの画面でも使える、共通なキー操作などは、ここにで定義を行う。       */
/*                                                                       */
/*************************************************************************/

void elKeyboard(void)
{
   //  [ESC]キーが押された場合……
   case VK_ESCAPE:
       //  このアプリケーションを終了させる。
       elDraw::Exit();
       break;

   //  elKeyboard関数の終了
   elExitKeyboard();
}

/*************************************************************************/
/*                                                                       */
/*  イベント関数                                                         */
/*                                                                       */
/*      ウィンドウズからの、何らかのイベントを処理したいとき、使用する。 */
/*  しかし、キーボードやマウス、MIDI監視など、ゲームに必要なほとんどのイベント */
/*  は、elが自動的に処理してくれるので、あまり活躍しない関数である。     */
/*                                                                       */
/*************************************************************************/

long elEvent(void)
{
  elExitEvent();
}

/*************************************************************************/
/*                                                                       */
/*  メイン画面関数                                                       */
/*                                                                       */
/*      先ほど定義した画面No (MAIN_SCREEN) と対になる、画面用の関数。    */
/*  ゲームらしいプログラムは、ここに書くことになる。                     */
/*  この関数がループで何度も呼び出されて、実際のゲームが進行して行く。   */
```

具体的にいうと，ここで定義されている数値38はプログラム中で頻繁に使用するX（横）座標の最大値だ。#defineを使用しないと，このX座標の定数を変更するたびに，その処理ごとに38という値を変更しなければならず，とても面倒な思いをするハメになる。

実のところ，定数の型（整数など）が決まっている場合はconstを使用するのが望ましいのだがそれはまたのちほど。

関数について

前章で，Cは関数の集まりでプログラムが構成されていると述べたが，ここで関数の取り扱いについて簡単に説明しよう。

void MainScreen（void）；

これは，プロトタイプ宣言と呼ばれるもので，プログラムで使用する関数をあらかじめコンパイラに「MainScreen関数を使いますよ」と伝えている部分だ。次に説明する変数と同じで，関数も事前に定義が必要なのだ。

プロトタイプ宣言した行からずっと下にあるvoid MainScreen（void）が，関数の本体で「{」から「}」までが，その関数の処理範囲である。

文の最後にセミコロン（;）がついているが，Cでは関数に限らず，文の終わりを示すためにセミコロンをつけなくてはならない。このように関数を呼び出したり，自分で関数を作ったりすることによってプログラムは構築されていくのだ。

今回の関数の使用法は，一般のCでの関数のそれとは少し違うので注意してほしい。

それとvoidは，値を持たないという意味だ。ここの部分は関数の値の引き渡しをするときに必要となるのだが，連載の初回ということで，特に意識しなくてよいようにサンプルプログラムを組んだ。混乱させて申しわけないが，これらのフォローは次号でする予定だ。

ちなみに本格的にコードを書き始めるのは，MainScreen関数以降だ。その以前の関数はelの実行に必要な処理で，ほとんど変更する必要はない。

次の説明に入る前にelの基本関数と処理の流れを解説しよう。プログラム中のコメントとあわせて読んで理解しておいてほしい（elLoopの処理部分の意味を大まかにとらえることができれば，プログラムの流れをつかめるだろう）。

・elMain

まず最初のelMainは，一般のC言語ではmain関数，WindowsでのWinMain関数にあたり，プログラムが起動すると最初に呼び出される関数である。

```
/*                                                                           */
/*****************************************************************************/

void MainScreen(void)
{
   // 画面が切り替わった直後の処理
   if (elChangeScreen())
   {
       // 初期化処理関数の呼び出し
       InitFunc();
   }

   // プレイヤー最新座標にヘビキャラクタを代入
   ImaginaryScreen[Px][Py]='@';

   // 仮想画面を表示
   elFont::Begin(GOTHIC,24);
   elFont::Color(RGB(255,255,255));

   for(i=0;i<MAX_Y;i++)
   {
       for(j=0;j<MAX_X;j++)
       {
           sprintf(Buffer,"%c",ImaginaryScreen[j][i]);
           elFont::Draw(j*17,i*17,Buffer);
       }
   }

   elFont::Before();

   // キー入力
   if (PushKey==VK_UP) D=0;       //「↑」
   if (PushKey==VK_DOWN) D=1;     //「↓」
   if (PushKey==VK_LEFT) D=2;     //「←」
   if (PushKey==VK_RIGHT) D=3;        //「→」

   // 方向によって座標を増減
   if (D==0) Py--;
   if (D==1) Py++;
   if (D==2) Px--;
   if (D==3) Px++;

   // ゲームオーバ?
   if (ImaginaryScreen[Px][Py]=='@'
       || ImaginaryScreen[Px][Py]=='#')
   {
       // ゲームオーバ画面処理関数へ
       elCallScreen(OVER_SCREEN);
   }

   // スコア処理
   Score++;
   if (Score>Hiscore) Hiscore=Score;

   // 画面を更新
   elDraw::Refresh();

}
```

```
/*****************************************************************************/
/*                                                                           */
/*  ゲームオーバ画面関数                                                       */
/*                                                                           */
/*****************************************************************************/

void OverScreen(void)
{
   elFont::Begin(GOTHIC,24);
   elFont::Color(RGB(255,0,0));

   // スコア表示
   sprintf(Buffer,"SCORE%5d HI-SCORE%5d",Score,Hiscore);
   elFont::Draw(180,200,Buffer);

   elFont::Draw(Px*17,Py*17,"X");
   elFont::Draw(234,230,"Hit Space key!");

   elFont::Before();
   elDraw::Refresh();

   if (PushKey==VK_SPACE)
   {
       // 再びメイン画面処理関数へ
       elCallScreen(MAIN_SCREEN);
   }

}

// ゲーム初期化関数
void InitFunc(void)
{
   // 画面をクリア
   elDraw::Clear();

   // 各変数を初期化
   Score=0;
   Px=rand()%30+5;              // Px座標をランダムに決定
   Py=3,D=1;

   // 仮想画面の初期化
   for(i=0;i<MAX_Y;i++)
   {
       for(j=0;j<MAX_X;j++)
       {
           ImaginaryScreen[j][i]='#';
       }
   }

   for(i=1;i<MAX_Y-1;i++)
   {
       for(j=1;j<MAX_X-1;j++)
       {
           ImaginaryScreen[j][i]=' ';
       }
   }

}
```

elMain後の()にプログラムのタイトルを入れる。

・elWindow

elWindowは，ウィンドウ表示を行う場合に使用する。最後のTRUEで，ウィンドウの外にマウスカーソルを出さないようにしている。このelWindowは，省略可能で，その場合はフルスクリーン表示となる。

・elLoop

elLoopは重要なループで，この中の処理が繰り返し呼び出されることによって実際のゲームが進行していく。

・elSetScreen

elSetScreenは，プログラム先頭の#defineで定義した画面No（MAIN_SCREEN,OVER_SCREEN）と，それに対になる画面用の関数（Main_Screen, Over_Screen）を定義している。つまり，elでは画面ごと（ゲームメイン部分，ゲームオーバ部分など）に関数を作成する。それをelCallScreen（後述）の画面指定によって，複数ある画面用関数のどれをelLoopで繰り返し呼び出すのかを決めている。

・elCreate

elCreateは，ウィンドウが生成される前に呼び出され，画面の解像度の設定を行う。この関数中のelCallScreenで，最初にどの画面を呼び出すかを決めている（サンプルプログラム中では，MAIN_SCREEN）。

・elKeyboard

elKeyboardでは，ゲーム全体に関係のある，つまり全イベントに共通なキーボード入力に利用する。たとえば，ESCキーでプログラムを終了させたいときに使うと便利だ。

以上，ざっと説明したが，elライブラリの詳しい使い方とel関数についてはヘルプを参照してほしい。¥EL¥eloperrateのドキュメントを一読すると，より理解が深まるだろう（そちらのほうが僕の上記の説明よりもわかりやすかったりする……）。なお，プログラム先頭のel関数処理のコメントは，elの作者であるBotchy氏のelヘルプを参考に，一部改変して引用させてもらった（多謝）。

変数

int Score=0;

変数とは要するに値が変化する「入れもの」で，これに文字や数値を入れたり出したりする（ゲームの点数，敵キャラの動き，画像データなど）。

変数を使う場合，前もって定義が必要で，先頭のintで「この変数は整数の数値を扱いますよ」と，コンパイラに伝えている。

intは，32ビット(-2147483648～2147483647まで，つまり約±21億)の

数値を扱うことができる型である。このデータ型についての説明は，表2にまとめたので詳しくはそちらを参照してほしい。

次のScoreというのが変数名（識別子）で，0を代入することによって値を初期化している。つまり，変数の定義と初期化を同時に行っているわけだ。

変数には好きな名前をつけることができる（英文字，数字，アンダースコア「_」）。名前は，できるだけその変数の実体がわかりやすいものにしよう。ただし，あらかじめ定義されている関数（予約語）や，変数の先頭が数字で始まるものは使用できないので注意。プログラム中で使われている主な変数は表3にまとめたので，軽く読んでおくといいだろう。

変数の適用範囲（スコープ）

変数には，有効な範囲というものがある。これを変数のスコープと呼ぶ。基本的には，関数内｛｝で括られた範囲がスコープとなる。プログラム先頭で宣言した変数はプログラムが終了するまで有効だが，ほかの関数内で定義した変数はその関数内でしか扱うことができない（今回のプログラムでは先頭で定義した変数しか使わないので，あまり意識しなくて済んでいるが……）。前者をグローバル（広域）変数，後者をローカル（狭域）変数と呼ぶ。

型修飾子　const

例：const int a=0;

簡単にいうと，一度constで定義された変数は，その値を変更することができない。いい換えれば，変数の値を定数扱いにするものだ。これ以降で変数aの内容を変更するようなプログラムを組み，コンパイルしようとするとエラーが発生するはずだ。

前出の#defineと同じような使い方をするが，型を明確にしたい場合こちらを使用するのが好ましい。演算などをするときには変数の型を揃える（これをキャストという）のが一般的だ（例：float型の変数とint型変数を演算すると予想外の値が返ってくる可能性がある）。説明の便宜上，今回はリストの先頭で#define MAX_X 38としたが，これは，

```
const int MAX_X=38;
```

としたほうがいいだろう。

今回は，キャストのことは特に意識しなくていい。これは，次号で解説する予定だ。

文字列と配列

charは主に文字を扱うときに使用する変数の型だ。

これはほかのデータ型にも共通することだが，変数の後ろに［］を加えることによって，ひとつの変数で指定した［］中の数値分＋1個の値（これを添字という）を持つことができるようになる。これが配列というものだ。

例：char a[3]="Hi!";

Cでは文字列型というものを持たない。文字列はchar型の配列で表現するのが一般的だ。この例の場合，最初のa［0］には，「H」，続いてa［1］は「i」，そしてa［2］には「!」が入ることになる。注意してほしいのは，配列は1ではなく，0から始まることと，文字列の場合には添字の宣言はひとつ多めにしてほしいということだ。なぜなら，C言語では文字列の長さの判定を簡略化するために（たいていの場合，宣言された配列のすべてが文字列で使われるわけではない），文字列の最後がどこまでであるかを示す記号として「Null（ヌル）」が必要なのだ（具体的には文字コード0x00）。Nullをわかりやすくいえば「この変数の文字列の値はここまでだよ」と終わりを教えてくれるものということになる。Cのプログラムだと，これがないと文字列がどこで終わるのかがわからず，最悪の場合，暴走につながる。

ということで，最後のa［3］には文字列の終わりを示す「¥0」（Null）が自動的に代入されている。つまり，配列要素数3で宣言されたchar型配列変数aで使用できる文字数は3つまでということになる。

ほかの型での配列も見てみよう。

例：　int a[3]={10,20,30};

これは，int型の配列の使用例だ。char型との違いはNullがいらないということだ。配列にまとめて値を代入するときにはこのような記述を行う。

では，プログラムに戻ろう。

```
char ImaginaryScreen[MAX_X][MAX_Y];
```

このも配列の例だが，添字の個数を増やすことによって，多次元配列を扱える。ここでは，前の#defineで定義された定数「MAX_X」(38)，［MAX_Y］(28)分の値を2次元配列として宣言している。

この2次元配列では，プログラム中で仮想画面として，キャラクター情報が保存されている。2次元配列の説明は仮想画面の取り扱いのところでまとめて行う。

演算

Cでの演算方法は，「+」で加算，「-」で減算，「*」で乗算（掛け算），「/」で除算（割り算），そして「%」で乗余算（割った余り）を行う。

例：　a=a+1;

この例では変数aに1を足した数が，再びaに代入されている。

また，上記のように1だけ足したり，引いたりしたい場合には，a++;（インクリメント）やa--;（デクリメント）がよく使われる。これは後述する繰り返しのforや，キャラクタの移動などによく利用される。

また＝の後の変数が共通な場合には，a+=1;とすることも可能である。

これらの演算は，状況に応じて使い分けるといいだろう。

条件分岐・if

さて，ここからいよいよ（やっと）ゲームのメイン処理であるMainScreen関数の説明に入る。

まずはifであるが，これはなんらかの条件が正しいかどうかを判定して，それらに応じて処理をしたい場合に使うものだ。

例：

```
  if (a==0) ←条件
   {
  処理
   }
```

もし変数aの値が0なら，条件が真（正しいということ）になり，if以後の｛｝で括られた処理が実行される。処理がひとつの場合，括弧を使わずにif文のすぐ後ろに書くことができるが，プログラムを見やすくするという意味では，あまり多用しないほうがいいだろう（変数の代入のような短い処理を除いて）。

例にあるとおり，その値が同じかどうか判定するには「＝＝」を使い，逆に違うことを判定する場合には「！＝」を使う。

次に，大小の判定であるが，これは単純に「＞」，「＜」でよい。注意してほしいのは「＞＝」，「＜＝」で，「＝」を含む場合には，大小記号の後ろに＝をつけることだ。

```
    if (elChangeScreen())
    {
  InitFunc();
       }
```

プログラム中では，ifの条件判断の部分で，elの関数elChangeScreenを使用している。

もし画面が切り替わった直後（つまり，ゲーム開始時）だったら，ifの条件が真となり，｛｝内のInitFunc();で，初期化に必要な処理を行う関数を呼び出している。

● InitFunc関数内の処理について

この関数は，プログラムのいちばん最後にあるのだが，この関数内で行っている重要な処理について説明しよう。

●裏画面と表画面

画面には裏画面と表画面の2種類があって，ディスプレイに表示されるのは表画面で，実際に絵を（文字を含む）描くところは裏画面である。

なんだかややこしい話だが，要するに裏側で絵を描き終わって，最後に裏画面と表画面を入れ換えれば，一瞬で絵が出てくるというわけだ。

●elクラス使用法

```
elDraw::Clear();
```

裏画面を黒で塗りつぶしなさいという（画面の初期化）elのクラス関数だ。これは，単純にelでの命令のようなものだと考えておいてもらえればいい。

elのクラスの使用法はいたって簡単で「C++の派生」や「継承」など，難しいことを考える必要はない。

目的に（画面の描画など）に応じて，「どのクラス内の（ここではelDraw）」の「どの関数（Clear）」を呼び出すのか，簡単に記述すればいいだけだ。

なお，実際に表画面表示させたいときは，elDraw::Refresh();だけでOKだ。

クラスというのは，C++で導入された概念なのだが，これは次号で詳しく説明する（次回への前ふりが多くて申しわけない……）。

前にも述べたが，elで提供されている命令（クラス）を詳しく知りたい場合はヘルプを参照のこと。

●乱数（でたらめな値）

```
Px=rand()%30+5;
```

関数の先頭部分で，ヘビの向き・座標変数の初期化をしているのだが，上記のように，ゲーム開始時のPx値（ヘビのX座標を保持している変数）を乱数で決定している。

rand();は乱数を発生させるための関数である。最初の「%30」で，最大30までの乱数を発生させ，5を足すことによって，結局5から35までの値を変数Pxに代入している。たいていのゲームは乱数を使用しているので，この関数の利用価値は，非常に高いといえるだろう*6。

*6　そういえば昔，乱数を一切使わない「ザナドゥ」というRPGがあった。このゲームのプログラマさんは「RPGに乱数など必要ない！」と豪語していたっけ……。これはまだ僕が，エロスを知らなかった小学生時代の，懐かしいセピア色の思い出だ。

●繰り返し・forループ

例：

```
for (初期値 ; forを繰り返す条件 ; 繰り返すためのカウント)
{
    処理
}
```

forは条件によって，一定回同じ処理を繰り返したい場合に使う。

```
for(i=0;i<MAX_Y;i++)
{
    処理
}
```

今回の場合，「変数iを0にして，変数iがMAX_Y（つまり28）より小さいあいだ変数iに1を加算して｛｝内の処理をしなさい」という意味になる。

繰り返すためのカウンタとして，「演算」で説明した1だけ変数の値を加えるインクリメントをしているのがわかるだろう。

プログラムでは，2つのループ（i,j）を使い，仮想画面の初期化（ゲーム開始時の画面設定）をしている。

●仮想画面

仮想画面とは，画面情報を2次元配列などによって保存する方法だ。これは，古くから使われている手法で，キャラクターの当たり判定を簡単にするなどのメリットがある。障害物や味方・敵キャラクターなどの情報を任意のX，Y座標を参照することによって簡単に得ることができる。

ここで，あと回しにしていた2次元配列の説明をしよう。

要するに，2次元配列ImaginaryScreen[Px][Py]は，添字の変数Pxが仮想画面のX座標，PyがY座標で，仮に以下の図に示すように，配列にキャラクターが入っていてPxに3，Pyに1が入っている場合には……。

図1
※2次元配列のイメージ図

```
  01234
0 ##################
1 #  @←ここ        #
2 #  @              #
3 #  @              #
4 #  @              #
  #                 #
  #                 #
  ##################
```

※ImaginaryScreen[Px][Py]の配列の中身は「@」が入っていることになる。

以上で初期化関数InitFuncの解説は終わりだ。

●sprintf

例：sprintf(a,"%d",b);

sprintfは変数の数値を文字配列に変換する関数だ。

上の例では，aという文字配列に変数bの数値を入れている。真ん中の「%d」は，変換する変数bの値を普通の数値（10進数）に直し，文字配列aに代入するという意味だ。

```
sprintf(Buffer,"SCORE%5d HI-SCORE%5d",Score,Hiscore);
```

これは，プログラムの最後のほうにあるゲームオーバー処理関数内のsprintf文だ。2番目と3番目のパラメータはひとつだけではなく複数取ることができ，文字列も入れることが可能だ。「%5d」で，5文字分右詰めをして，10進数変換をしている。

なお，自動的にBuffer配列の最後にはNullが入る。

「%」で始まる変換指示記号は，図2にまとめたので参考にしてほしい。

さて，プログラムの解説に戻ろう。

```
sprintf(Buffer,"%c",ImaginaryScreen[j][i]);
```

前にある2つのforループで仮想画面の内容を更新して，次に説明するelFontで画面表示を行っている。2番目のパラメータ「%c」で文字コードに変換している。3番目のパラメータで，ImaginaryScreen[j][i]を指定しているが，これは仮想画面の内容を変数Bufferに代入している部分だ。

最初のループで仮想画面のY座標，次のループでX座標を足している。

図2　変換指示記号

%d：10進数変換
%x：16進数変換
%u：符号なし10進数変換
%c：1文字変換
%s：文字列変換。ただし，必ず最後にNullを含んでいないといけない

表2　数値のデータ型と扱える範囲

・char：-128～127（8ビットの整数）
・short：-32768～32767（16ビットの整数）
・long：-2147483648～2147483647（32ビットの整数）
・int：現在は32ビット（つまりlongと同じ）が主流
※データ型の前にunsigned（符号なし，正の数のみ使用）をつけることによって，扱える数値は2倍に増える
・floatは，32ビットの小数
・doubleは，64ビットの小数
これは，浮動小数点数と呼ばれ，実数を扱いたい場合に使用する

表3　ゲーム中の主な変数

Score,Hiscore……得点関連
ImaginaryScreen……仮想画面
D,Px,Py……ヘビの向きとX,Y座標
PushKey……キー入力（elの公開変数）

図3　ゲームで主に使用するWin32 APIの仮想キーコード一覧

VK_UP,VK_DOWN,VK._LEFT,VK_RIGHT…..カーソルキー
VK_ESCAPE……エスケープキー，VK_SPACE……スペースキー
VK_RETURN……エンターキー
VK_NUMPAD0～VK_NUMPAD9……テンキー
※ 一般のASCIIコードで表現できるAや1は，そのまま'A'と'1'というふうに使える。詳細はWin32APIのヘルプを参照のこと

● 文字表示

```
elFont::Begin(GOTHIC,24);
elFont::Color(RGB(255,255,255));
```

elFontは文字を表示するためのel関数だ。Beginでフォント（文字），大きさを指定し，Colorで色を決める（ここでは白）。

```
elFont::Draw(j*17,i*17,Buffer);
```

j,iは表示座標で，変数Bufferの内容を表示している。なぜ17をかけているかというと，表示する文字の間隔がそれぞれ17ドットの大きさだからだ。

```
elFont::Before();
```

で文字表示の終了だ。

● キーボード入力とキャラクタの移動

```
if (PushKey==VK_UP) D=0;
if (PushKey==VK_DOWN) D=1;
if (PushKey==VK_LEFT) D=2;
if (PushKey==VK_RIGHT) D=3;
```

キーボード入力は，el公開変数のPushKeyで行っている。

この変数にはキー入力の状態が保存されているので，if文でその変数の内容をチェックするだけで，キーの判定ができる。

「VK_UP,VK_DOWN,VK_LEFT,VK_RIGHT」は，仮想キーコードと呼ばれているもので，それぞれのカーソルキーに対応している。仮想キーコードについては，図3を参照。

ここのif文でキー入力状態に応じ[*7]，ヘビの進行方向（変数D）を決定している。

*7　本当はここで，if-elseを使うべきなのだが，諸般の事情により（？），if文オンリーにさせてもらった。ま，簡単に修正できる部分なので，自分で書き換えてみるのもいい勉強になるかと……。

●もう１つの条件分岐・switch

```
if (D==0) Py--;
if (D==1) Py++;
if (D==2) Px--;
if (D==3) Px++;
```

進行方向に応じて，ヘビのX・Y座標を増減している。

ここの部分はswitchで記述することもできる。特に何種類かの選択肢があり，そのなかからひとつを選ぶ……という処理をしたい場合に便利だ。さて，実際に置き換えてみよう。

```
switch(D)
{
case 0:
    Py--;
  break;

case 1:
  Py++;
  break;

case 2:
    Px--;
  break;

case 3:
  Px++;
  break;

  default:
  break;
}
```

変数Dの値が0の場合case 0:が実行され，Pyの値が1引かれて，break:でswitch文の処理から抜ける。case1以降も同様である。ただ，最後にどの条件にもあわない場合にdefault:が実行される。

●ゲームオーバー？

```
if (ImaginaryScreen[Px][Py]=='@'
 || ImaginaryScreen[Px][Py]=='#')
```

ここでは単純に仮想画面のプレイヤー現在座標を確認して，もし自分の体「@」や壁「#」がある場合には，ゲーム終了の処理を行うゲームオーバー関数OverScreenを呼び出している。

ここのif文で注意してほしいのは，「||」の部分だ。これは論理演算子と呼ばれるもののひとつで，「||」はORで，式の両方ひとつでも正しければ真になる。ANDは「&&」で，式の両方，NOTは「!」でどちらも正しくないときに真となる。

●スコア処理と画面の更新

```
Score++;
if (Score>Hiscore) Hiscore=Score;
```

スコア処理で，1回ヘビの体が伸びるごとに，変数Scoreの値を1足している。もしハイスコアが更新されたのなら，Scoreの値がHiscoreに代入される。

```
elDraw::Refresh();
```

最後に，いままでいろいろと描画してきた裏画面を実際にディスプレィに表示される表画面に切り替えて，再びMain_Screen関数の最初から処理が繰り返される。

●OverScreen関数

ここでは単純にミスした場所に赤く「X」を表示し，スコア表示を行っている。スペースキーが押されると，elCallScreenでMainScreen関数が再び呼び出されて，ゲームが始まる。

以上で，今回のサンプルプログラム〝Snaky1〞の解説は終わりだ。読者の皆さん（自分にも）お疲れさま……。

そして次号へ……(Say hello, and good bye.)

今回は，ゲームを作るために最低限必要な事項を説明した。C言語とはどんなものか大まかにとらえてもらうために，どうも説明をはしょりすぎた感があるが，あとは各自，必要に応じてオンラインヘルプや，Cの文法書なりを参照してほしい。

次回では，Snakyの2プレイヤーバージョンをサンプルに「関数の扱い」，「構造体」，「キャスト演算子」，「ポインタの基礎」，「クラスの概念」，「機種に依存しないゲーム速度」などの解説をしたい。

それに加え，今回のサンプルはピコピコゲームとは名ばかりで，音を出す段階までたどり着けなかったので，その辺もきちんと対処したいと考えている。

Oh! X春号で広井誠氏によって解説されたTcl/Tkは，手軽にゲームを作れる優れた手段のひとつだ。しかし，ほかにも選択肢があっていいはずだ。

この連載の目的は「意外とCで簡単にゲームを作れるんだな」と感じてもらうことである。そして，今度はあなた自身がゲームを作り，その作品をOh!Xに投稿をしてもらえれば最高だと思う。

では，次号の誌面でまたお会いしよう。

Webページでグラフを表示しようⅠ

大和 哲　Yamato Satoshi

特集がグラフィック関係ということで，Webページでもグラフィカルな表現をCGIで実現してみよう。ここでは手始めに，インタラティブなグラフ作成について見ていこう。正確にはグラフィック処理ではないのだが，比較的用途の多いリストだ。

CGIでグラフを描こう！

さて，復刊1号からCGIを使って（でも，よくあるCGIプログラムにはあまりないようなものをちょっとだけ目指して）いろいろなことをしてみているこのページなのですが，今回はWebページ上でグラフを表示してみています。しかも同じ号の同じ雑誌の中でブラウザにグラフを表示させる記事が2つ掲載されています。ひとつはこちらの「Webページでグラフを表示しようⅠ」，もうひとつは「Webページでグラフを表示しようⅡ」です。どちらかというと，こちらのⅠのほうは，普通にCGIに使われている（つまりHTMLテキストを出力する）方法でプログラムを作りますので，プロバイダでCGIを使える環境ならば誰でも使うことができます。CGIを初めて使う方でも使用方法に従えば，プログラムを使うことだけはできるように心がけています。もっとも手軽な使い方としては「サンプルプログラムの使い方」を読むだけで，ご利用になっているWebサーバにサンプルプログラムを転送して実際に使うこともできるかもしれません（多少は書き換える必要のある場合が多いと思うのですが……）。

逆に，Ⅱのほうは，telnetでシェルログインしてプログラムがコンパイルするなど，かなり特殊な作業を必要とするので使える人は限られてしまうかもしれません。

さて，こちらのⅠのほうで作るサンプルプログラムとして，「アンケート投票棒グラフ」を作ります。まず，htmlでアンケートページがあり，そこでお好きな項目に1票を投じると，現在までの結果がグラフで表示されます。

この棒グラフを表示する，というところがミソなわけですね。でも，こんなグラフをどうやって表示するのでしょう。

●CGIとは

グラフの出し方を説明する前に，CGIとはどのようなものかを簡単に復習しておきましょう。ブラウザでホームページを見るときには，アドレス欄の，

http://www.foo.com/~bar

というように「http」リクエストをブラウザがサーバに対して出します。このhttpリクエストでは，普通は，Webサーバがhtmlファイルやgifファイルをそのままブラウザに返します。CGIは「Common Gateway Interface」といって，このデータの通り道にいるプログラムです。CGIを呼び出すと，サーバ中にあるファイルをそのままブラウザに返すのではなくて，このプログラムがhtmlファイルなどを加工して出力するわけです。

CGIプログラムはWebサーバから制御されているわけですが，プログラムがCGIとして起動されると，CGIプログラムの標準出力がそのままネットワークへの出力，つまり，ブラウザが受け取るデータということになります。つまり，ブラウザにHTMLで書いたようなページを表示させたければ，ブラウザに「これはHTMLデータだよ」ということを教えてやればいいわけですね。そこで（CGIを一度でも作ったことのある方なら見たことがあると思いますが），CGIプログラムでは，まず何より先に，

```
print "Content-Type: text/html¥n¥n";
```

と，ブラウザに「これから流すデータはHTMLのデータだよ」という宣言を出力します。

あとは，

```
print "<HTML>…
```

と，いつもホームページを作るときに書いているHTMLの内容をprint文などで出力するようにすればブラウザ上に，可変的なホームページデータを表示させることができるわけですね。

HTMLでグラフを作ってみせる

さて，CGIではHTMLテキストをWebクライアントに流すことができるわけですので，ここでは，HTMLとそこで表示できる画像ファイルを使って棒グラフを表示する方法を考えます。

HTMLの文法はW3Cという組織で決められていて，ホームページでその仕様が公開されています。HTMLでグラフィックイメージを扱うのは<IMG>タグですので，Web上で，このタグの仕様を調べてみましょう。HTML4.0でのIMGタグの詳細はここにあります。

http://www.w3.org/TR/REC-html40/struct/objects.html#h-13.2

日本語訳はこちらです。

http://www.asahi-net.or.jp/~bd9y-ktu/html4rec_f/struct/objects.html　#h-13.2

これによるとIMGタグは以下のように書かれます。

```
<IMG SRC=%Uri alt=%Text longdesc=%URI height=%Lengthwidth
=%Length usemap=%URI ismap         (ismap) >
```

SRC=のあとにURLを書くことで，そのURLにあるイメージを表示するわけですね。alt=" "で画像が表示されなかった場合に適当な文字列を表示します。

さて，ここで，widthとheightというパラメータに注目してください。よく，これらは「Webでの表示スピードを速くするために」画像のピクセル数をここに入れたりします（ブラウザは通常，イメージデータをダウンロードできるまで画像の大きさがわからないので，ページの中にどのように文字や絵を配置するかと計算する際に時間がかかってしまうのですが，この数値を使えるとその分ページを速く表示することができるのです）が，これらはイメージを表示する際に大きさを変更するためのパラメータで，画像の本来のピクセル数以外の数値を入れると，画像を大きくしたり小さくしたりして表示することができるのです。HTMLファイルをエディタなどで書いて試してみてください。本来128×128の画像であっても，

```
<IMG foo.gif width=64 height=128>
```

と書けば縦が半分に，

```
<IMG foo.gif width=64 height=256>
```

と書けばその画像をさらに縦に2倍に引き伸ばすことができます（図1）。単純にピクセルが引き伸ばされるだけなので，画像は汚くなってしまいますが。

このタグを使って棒グラフを描いてみることにしましょう。理屈はこうです。まず，適当な大きさのGIFファイル（仮にbar.gif）があるとしましょう。グラフは下のように，

```
項目1 | ■■■■■■■■■■■■■■■■■
項目2 | ■■■■■■■■■■■■
項目3 | ■■■■■■■■
項目4 | ■■■
項目5 | ■
```

図1

各項目が縦に並び，それぞれの量を示す棒が横に伸びるものとします。この横に伸びるグラフをGIFイメージと<IMG>タグで操作します。仮に棒の太さが24ピクセル，n%がddピクセルであるとして，そのグラフを表示したいのであれば，CGIプログラムは次のようにHTMLテキストを出力すればいいわけですね。

```
<IMG SRC=bar.gif width=dd height=24>
```

拡大するとGIFファイルが汚くなってしまいますが，GIFファイルが単色で塗りつぶされているだけなら問題はないですね。

ただ，ひとつ，この方法では見た目の問題としてブラウザによってはwidth=0の場合でも，1ピクセルだけ棒が描かれてしまうブラウザがあります。このような見た目の問題は，各ブラウザの問題なので残念ながらあきらめるしかありません。とはいえ，度数100を400ピクセルや600ピクセルなどの大きなドット数で表現するならば，度数0と1の違いは見てわかりますので，許していただきましょう。

そうそう，w3mやlynxのように画像を見ることのできないブラウザのために，ALTタグでパーセンテージをテキストで書いておくとより親切かもしれませんね。

データの持ち方

サーバ側でのデータの持ち方は簡単です。

Perlには「ハッシュ」といって，配列の添え字に文字列なども含めてスカラー全般を使うことのできる配列があります。普通の配列では，

```
$array[1]=first;
$array[5]=fifth;
```

とすると，@array配列の1番目と5番目にfirst，fifthという文字列が代入されますが，ハッシュでも同じようにキー(配列での添字のことをハッシュではキーと呼びます)にスカラーで，

```
$hash{"Windows9x"}=25000;
$hash{"Linux"}='Free!';
$hash{"WindowsNT"}='高い!';
$hash{"FreeBSD"}=0;
```

Windows9x	25000
Linux	Free !
WindowsNT	高い！
FreeBSD	0

のように代入すると，%hashハッシュには以下のように値が代入されるわけです。

ただし，通常の配列と違って，内容の配置は順不同なので，配列がどのような順番に入っているかを前提に配列の0番目から5番目を表示……というようなプログラムを組んではいけません(ハッシュの要素をすべて列挙したり，ソートすることはもちろんできます)。もちろん，ハッシュのキー，内容とも，数値を入れることもできます。JavaScriptでは配列の添字に数字，文字，どちらでも使うことができますが，Perlにはあのタイプの配列(ハッシュ)と通常の数字だけの添字の配列があると思えばだいたいあっているでしょう。

各アンケートの項目もハッシュで，各項目とその値を持つとすっきりできますね。

なお，このハッシュはそのままdbmファイルというものに結び付けて，そのままディスクにファイルとして残しておいたり，あるいは読み込んでおくこともできます。Perl5のtieという関数がそれです。

```
tie %ハッシュ名,DB_File,ファイル名,O_RDWR,0666,$DB_HASH
```

これでハッシュを読み書き可能でDBMファイルに結び付けることができます。たとえば，

```
tie%surveydata,DB_File,"$homedir/surveydat.db
m",O_RDWR,0666,$DB
```

とすると，%surveyというハッシュが$homedir/surveydat.dbmというDBMファイルに結び付けられます。ここで，

```
$survey{"Windows9x"}=25000;
```

と値を代入すると，DBMファイルの"Windows9x"が指す部分も，

Windows9x	25000

と書き換えられます。

そしてハッシュとDBMファイルの結びつきはuntie文で切り離すことができます。

```
untie %survey
```

で%surveyハッシュに結び付けられていたDBMファイル，surveydat.dbmが切り離されます。

この時点では%surveyも，

Windows9x	25000

surveydat.dbmファイルの中でも，

Windows9x	25000

のように記録されているわけですが，このあと，

```
$survey{"Windows9x"}=12800;
```

と値を代入すると，%surveyハッシュの中は，

Windows9x	12800

ディスク上のsurveydat.dbmは，

Windows9x	25000

のまま残るわけです。

ホームページからのデータの取り方

さて，なにはともあれ，アンケートページなのですから，データの格納，表示だけでなく，ページを見ている人からアンケートを取って，そのデータをCGIプログラムが手に入れなくてはなりませんね。ページ上のデータをサーバ側のプログラムが手に入れるにはいろいろありますが，アンケートCGIなどの場合，よく使われているのが「フォームを使う方法」です。たとえば，この記事のサンプルページのように，ホームページ上に，

お好きなひとつを選んでください

[ ]DOS
[ ]Windows95
[ ]Windows98

［送信］

というように選択項目があり，その下の「送信」ボタンをクリックすると答えられるページというのを見たことがある方も多いでしょう。このページはHTMLの<FORM>タグを使って書かれているのです。

このFROMが書かれたWebページで，該当する項目をクリックし，「送信ボタン」をクリックすると，FORM文中で指定されたURLのCGIプログラムが実行され，CGIがパラメータを読み込んでグラフを表示します。つまり，アンケートCGIを実行するには，ファイルとしては，

図2

・FORM文でアンケート項目が書かれたHTMLファイル…(1)
・アンケートの結果を受け取って表示するCGIファイル…(2)
・これまでのアンケート結果が入ったDBMファイル…(3)

の3つが必要になるわけです。

そして，時系列に従うと，

1:「人1」が(1)HTMLファイルをブラウザで見ます
2:表示されたHTMLファイルの必要な項目を埋めて，「人1」が「送信」ボタンをクリックします
3:ブラウザによって指定されたCGIプログラムが起動します
4:CGIプログラム(2)2:で記入されたアンケート項目を見に行きます
5:CGIプログラム(2)はアンケートの結果に4:で受け取った2:の答えをDBMファイル(3)追加します
6:CGIプログラムは，アンケートの今までの状況を出力します
7:「人1」がブラウザ上でCGIの出力した結果を見ます

HTMLを書く

さて，HTML文の<FORM>の内容をどのように書くのか，具体的に見ていきましょう。アンケートページsurvey.htmlでは以下のように書かれています。

```
<FORM ACTION="survey.cgi" METHOD=GET>
<INPUT TYPE=RADIO NAME="ANSWER" VALUE="DOS">DOS<BR>
<INPUT TYPE=RADIO NAME="ANSWER"VALUE="Windows95">
Windows95<BR>
<INPUT TYPE=RADIONAME="ANSWER"VALUE="Windows98">
Windows98<BR>
(中略)
<INPUT TYPE=SUBMIT VALUE="送信">
</FORM>
```

<FORM>タグ中のACTION=は，このページで送信ボタンが押された場合に，このACTION=で指定されたURLにあるファイルが実行されます。つまり，ACTION=survey.cgiであれば，送信ボタンがクリックされた場合には，ここで指定されたsurvey.cgiが起動されます。METHOD=ではCGIプログラムにどのような形でアンケート結果を渡すのかの指定をしています（正確にはCGIにどのメソッドでリクエストをするかなのですが……。とりあえずはこう覚えていても問題はないでしょう）。簡単にいうと，このメソッドにはGET，POSTがあります（HEAD……と考えた方は考えすぎです。データがCGIに渡らないんだから使えませんね）。GETではデータの取り方が簡単ですが，ある程度以上長いデータは送れません。つまり，自由に長い文章を書いて送らせたいような用途には不向きです。こういうときには長いデータでも問題なく取得できるPOSTを使うべきですが，一から自分でデータを取るCGIプログラムを作ろうとすると面倒でしょう。幸い，いまはCGIを作るためのPerlのモジュールがたくさん配布されており，それらを使えばその辺の手間は全然かからないので問題ないのですが。今回は受け取るパラメータも短いのでGETを使ってしまいます。

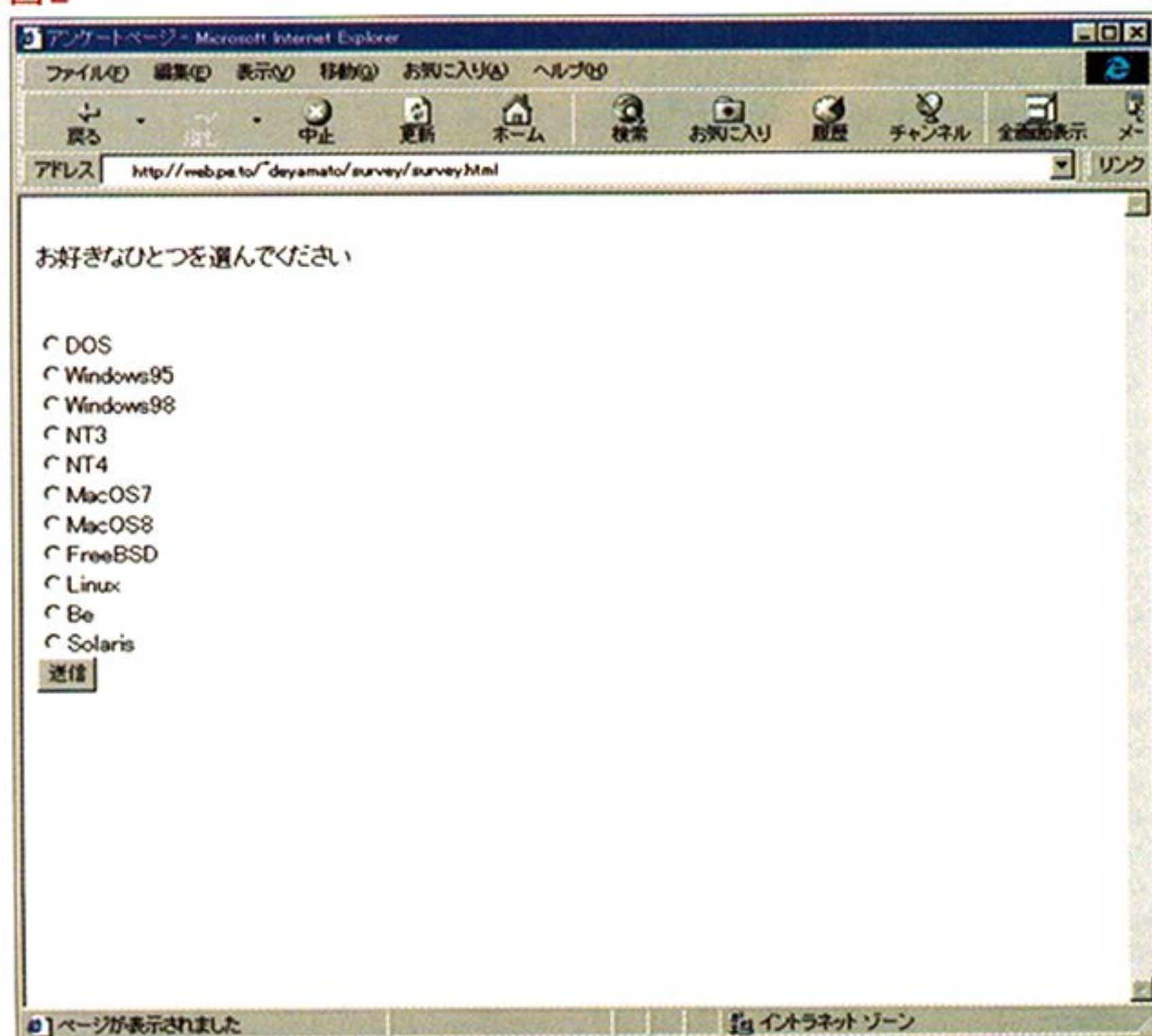
続いての<INPUT>タグが質問の1項目になります。

```
<INPUT TYPE=RADIO NAME="ANSWER" VALUE="DOS">
```

と書かれていた場合，CGIプログラムにはANSWERという変数に「DOS」という値が入って渡されます。このANSWERの部分を好きな文字列にすれば，好きな項目名をCGIに送ることができます。

```
<INPUT TYPE=SUBMIT VALUE="送信">
```

これが「送信」ボタンを表示するためのINPUTタグです。これとFORMの終わりを表す</FORM>は常にセットで覚えていてもかまわないと思いますが。

HTML側はだいたい，この程度書いておけば大丈夫でしょう。あとは注意事項としては，CGIで使う文字コードとあわせて同じ文字コードで書いておくことくらい注意しておけば大丈夫でしょうか。今回のサンプルとしてCD-ROMに収録しているプログラムは簡単にするために文字コードをすべてEUCで扱っています。ですので，アンケートページサンプルsurvey.html

もEUCで書かれています。ブラウザでこのページを見て，アンケートに答えてCGIに送られる文字コードももちろんEUCで送られるわけです。

なぜEUCで受け渡し，内部処理をするかというと，端的にいうと，シフトJISでは通常使う半角英数字のうちの一部（たとえば「¥」など）が漢字で使うコードと重複してしまうために処理が面倒になるからです。EUCでは，半角英数字は$7F以下，全角文字で使われるコードは必ず$80以上になりますので，日本語を考慮していない処理系でも操作に誤りが出にくいのです。特に，ハッシュを使うときにはプログラム内部での変数の管理やキーの検索などに問題が出ることもありえますので，もし，どうしてもHTML側をシフトJISなどで書きたいという場合は，CGI側で受け取ったパラメータをjcode.plでEUCに一度変換して使用することをおすすめします。

そしてCGI側

さて，最後にCGIプログラムのFORMデータの受け取り方も簡単に解説しておきましょう。

FORMで入力された値，たとえば，

```
<INPUT TYPE=RADIO NAME="ANSWER" VALUE="DOS">
```

というようなINPUTタグで表示されるボタンがチェックされていた場合，

```
ANSWER=DOS
```

というような文字列が渡されます。CGIプログラムから見るとMETHODにGETを指定した場合には，クエリー文字列という環境変数に，POSTメソッドの場合は標準入力にこの文字列が入っていますのでreadで文字列を読み取ります。

あとは，「=」で式が書かれているので，

```
split/=/
```

でその文字列を分けてやれば，パラメータの名前と内容を知ることができるわけですね。

さて，サンプルプログラムでは，この辺の処理はcgi-lib.plというcgi用のモジュールを使って処理しています。14行目の，

```
&ReadParse(*input);
```

で$inputにクエリー文字列が代入されているわけです。

このcgi-lib.pl（は，実はちょっともう古いんですが……）をはじめとして，PerlにはCGIのスクリプト，モジュールがいろいろと配布されています。1から作るのも悪くはないですが，このような定型的な部分はすでにあるモジュールを流用すると手間が省けてとても便利です。あるモノはできるだけ利用させていただいて，もっと面白いCGIの使い方のアイデアのほうに注力する，のが限りあるひとりの力でなにかをやろうとするのであれば，やっぱりいいですよね。本人も楽しいし。

なお，このサンプルプログラムでは，もともとDBMファイル（surveydat.dbm）になかった項目は不正な投票として「その項目では投票できません」と表示して終了させるようになっています。これは，たとえば，このサンプルプログラムがINPUTタグにNAMEで書かれていた文字列をそのまま投票の項目として使っているためです。HTMLファイルがCGIをACTIONで呼び出すような構造になっているので，このアンケートページにない項目でも投票しようと思えばできてしまうからです。たとえば，このsurvey.cgiを呼び出すようなHTMLファイルを別の場所に書いてその中のINPUTタグで不正な文字列を書いておくと，それがそのまま投票できてしまいますね。ですので，あらかじめアンケート項目はそれに対応した項目が書かれたdbmファイルを用意しておかなくてはなりません。もし，自分でそのようなdbmファイルを作れないとか，不正な投票でも許したいなどという場合にはsurvey.cgiの46行目から56行目までの行の先頭に#をつけて注釈にしてしまってください。ただし，ここに注釈をつけるとどんな投票でも受け付けてしまうことになりますので気をつけてください。

サンプルプログラムの使い方

このサンプルプログラムは，簡単なアンケートページとその集計CGIプログラムです。ページを見た人はそのページ中に表示された選択肢の中からお好きなひとつをクリックし，「送信」ボタンをクリックします。すると，これまでに何人の人がどの項目を選んだかが棒グラフとともに表示されます（項目は投票した人数の多い順にソートされて表示されます）。

このCGIの実行に必要なのはPerl5が使える（もちろんCGIが使える）Webサーバ環境，そして，下に書かれている付録CD-ROMに収録されているプログラムとデータになります。

initsurvey.pl（DBMファイル初期化プログラム）
survey.html（アンケートページ）
survey.cgi（アンケートCGI）
surveydat.dbm（アンケートデータ）
enum.txt（項目サンプルファイル）

まずは，enum.txtの要領でアンケートを取りたい項目を列挙してください（改行コードに注意。LFのみです）。続いて，これらのファイルのうちパス名など，変更が必要な部分を修正してください。このサンプルプログラムでは，Perl5のパスは/usr/local/bin/perl，CGIプログラムとHTMLデータとは同じディレクトリに入っていると仮定されています。Perl5のパス名はinitsurvey.plとsurvey.cgiの先頭に，CGIとHTMLファイルの位置の関係はsurvey.htmlの9行目「ACTION="survey.cgi"」に書かれています。それぞれ，たとえば，Perl5のパスが/usr/bin/perlであればinitsurvey.plとsurvey.cgiの先頭行を，

```
#/usr/bin/perl
```

に，CGIファイルがcgi-binディレクトリにあるのでしたら，survey.htmlの9行目は，

```
ACTION="cgi-bin/survey.cgi"
```

に書き換えてください。

次に，これらのファイルをサーバに転送します。FTPツールなどでファイルをサーバに転送してください。CGIを実行する際にはたいていのサーバでパーミッションの設定が必要になりますのでその作業も一緒に行うといいでしょう（なかにはinfoWebのように，転送後自動的にパーミッション設定をサーバがする場合もあります。その場合は必要ありません）。パーミッションの設定方法は各サーバ（プロバイダ）によって解説されていますのでそちらを参照しましょう。一般的には，telnetでログインしてシェルからか，あるいはFTPソフトで実行ファイルは，

```
chmod 755 ファイル名
```

読み書きするデータファイルは，

```
chmod 666 ファイル名
```

で設定することが多いはずです。サンプルプログラムの場合，

リスト1

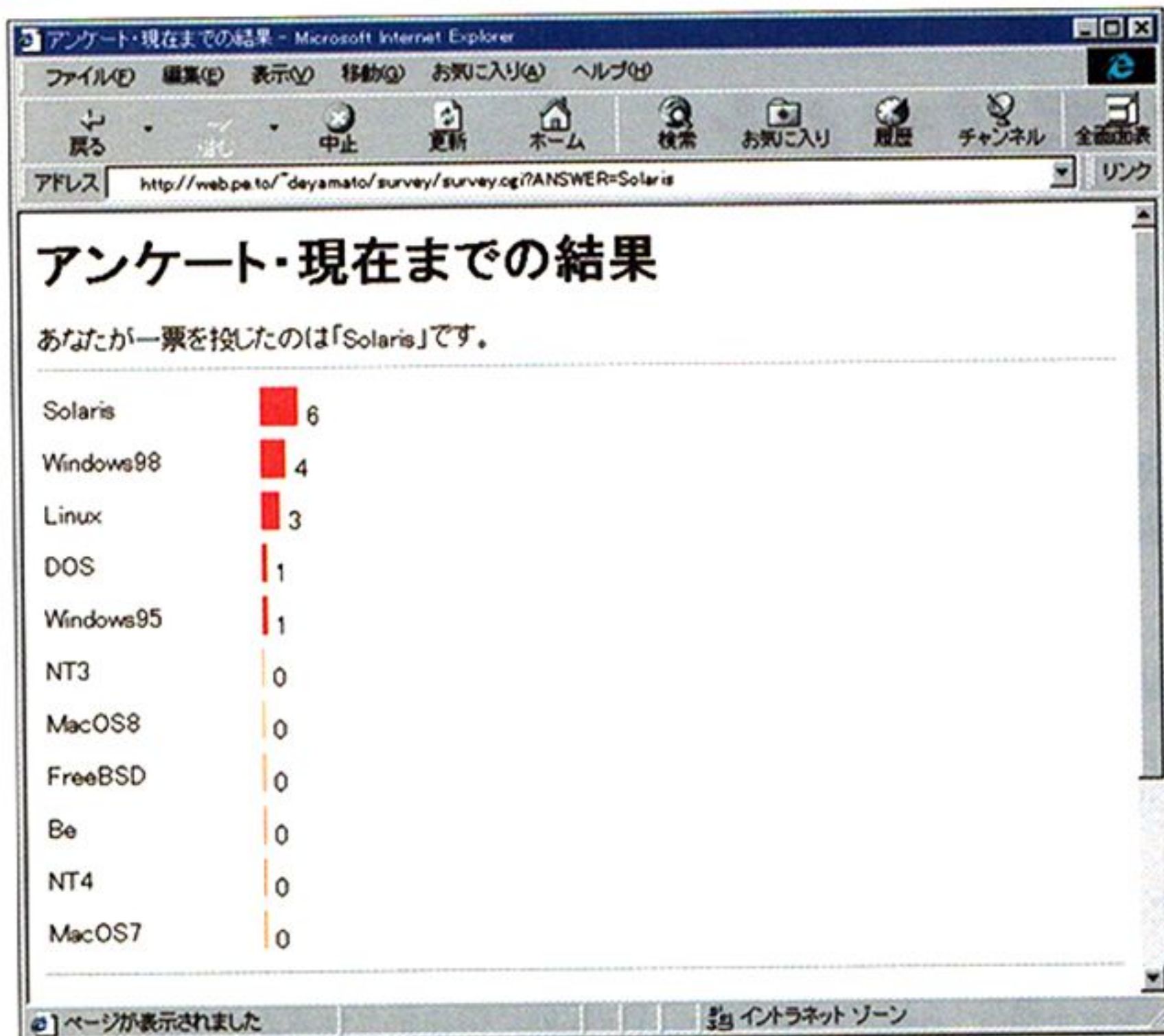
```
<HTML>
<HEAD>
<TITLE>アンケートページ</TITLE>
<META NAME=GENERATOR CONTENT=" HOTALL Ver.5.0W" >
</HEAD>
<BODY>
<P>
<BR>
お好きなひとつを選んでください<BR></P><FORM ACTION=" survey.cgi"
METHOD=GET>
<P>
<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" DOS" >DOS<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" Windows95" >Windows95<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" Windows98" >Windows98<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" NT3" >NT3<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" NT4" >NT4<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" MacOS7" >MacOS7<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" MacOS8" >MacOS8<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" FreeBSD" >FreeBSD<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" Linux" >Linux<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" Be" >Be<BR>
<INPUT TYPE=RADIO NAME=" ANSWER"  VALUE=" Solaris" >Solaris<BR>
<INPUT TYPE=SUBMIT VALUE=" 送信" ><BR>
<BR>
<BR>
<BR></P>
</FORM><P>
<BR>
<BR></P></BODY>
</HTML>
```

図3

アンケート・現在までの結果 - Microsoft Internet Explorer
http://web.pe.to/~deyamato/survey/survey.cgi?ANSWER=Solaris

アンケート・現在までの結果

あなたが一票を投じたのは「Solaris」です。

Solaris	6
Windows98	4
Linux	3
DOS	1
Windows95	1
NT3	0
MacOS8	0
FreeBSD	0
Be	0
NT4	0
MacOS7	0

　initsurvey.pl
　survey.cgi
が実行ファイル，
　surveydat.dbm(アンケートデータ)
がcgiの使うデータファイルです。

さて，続いてはenum.txtファイルのデータから，surveydat.dbmにアンケート項目の初期値を登録します。お使いのWebサーバでシェルが使える場合telnetでサーバにログインし，

　./initsurvey.pl <enum.txt

としてください。

この作業はPerl5さえあればできますので，もし，Webサーバにtelnetでログインできなくて，手元にPerl5が動く環境がある場合，そちらで実行してもいいでしょう(Linuxマシンなどがあるとcgiをローカルでテストできますので，作っておくと便利ですね)。画面(標準出力)にはこの項目を使用した場合，survey.htmlにどのようにINPUTタグを書くか，そのサンプルが表示されますので，コピーしてsurvey.htmlを書き換えるといいでしょう。

これでセットアップ作業は完了です。

動作を確認するために，ブラウザでsurvey.htmlを表示して，どれかひとつ項目を選んで「送信」ボタンをクリックしてください。選んだ項目に1票が投じられて集計結果のグラフが表示されます。

リスト2

```
#!/usr/local/bin/perl

#Form Test
#       by deyamato 1999

require “cgi-lib.pl”;
require “jcode.pl”;
use DB_File;
my @val;

my $homedir=”/home/deyamato/public_html/survey”;
my $answer_tag=”ANSWER”;

&ReadParse(*input);

print &PrintHeader;
print’<META content=”text/html; charset=charset=x-euc-jp” http-
equiv=Content-Type>’;
print &HtmlTop(“アンケート・現在までの結果“);

@val = split(/&/,$input);

foreach $i(0 .. $#val){
        $val[$i] =~ s/%(..)/pack(“c”,hex($1))/ge;
        ($name,$value) = split(/=/,$val[$i],2);
        $value =~ s/\+/ /g;
        $val{$name} = $value;
}

foreach $name(sort keys(%val)){
        $val{$name} =~ s/\n/”<BR>\n”/ge;
         printf &jcode’euc(“あなたが一票を投じたのは「$val{$name}」です。
<BR>\n”);
}

my $title=&jcode’euc($val{$answer_tag});

if($title){
        if( -e “$homedir/surveydat.dbm”){
                                                tie  %surveydata,DB_File,”
$homedir/surveydat.dbm”,O_RDWR,0666,$DB_HASH
                || &FileOpenError();
        }else{
                                                tie  %surveydata,DB_File,”
$homedir/surveydat.dbm”,O_CREAT|O_RDWR,0666,$DB_HASH
                || &FileOpenError();
        }

        # -- 不正投票チェック(dbmファイルに既にない項目なら不正とみなす)
        my $exist=0;
        while(($key,$dummy) = each %surveydata){
                if(($key cmp $title)==0){
                        $exist=1;
                        last;
                }
        }
        if($exist==0){
                print jcode’euc(“その項目では投票できません。\n”);
                print &HtmlBot;
                exit(0);
        }
        # --



        #-- ファイルに1書く
        $surveydata{$title}++;
        #print “$title = $surveydata{$title} <BR>\n”;
        #-- endof ファイル書き込み
        untie %surveydata;
}

print “<HR>\n”;
# -- 表示
print “<TABLE WIDTH=100%>\n”;

tie  %surveydata,’DB_File’,” $homedir/surveydat.dbm” ,O_RDWR  ||
&FileOpenError();;
foreach $key (sort HashValueCmp keys %surveydata){
        $i = $i +1;

                print “<TR><TD WIDTH=20%>\n”;
        print “$key”;
                print “<\/TD><TD VALIGN=MIDDLE>\n”;
                my $width=$surveydata{$key}*4;
                    print “<IMG SRC=\” basicbar.gif\”  ALIGN=BOTTOM
WIDTH=$width Height=24>\n”;
                print “$surveydata{$key}\n”;
                print”<\/TD><\/TR>\n”;
}
untie %surveydata;
print “</TABLE>\n”;
# -- endof 表示
print “<HR>\n”;
print”<BR>\n”;

print &HtmlBot;
exit(0);
# end

# 応答コードの異常の場合
sub FileOpenError
{
        print “ファイルオープンエラー発生しました.<BR>”;
        print &HtmlBot;
        exit(0);
}

sub HashValueCmp
{
        $surveydata{$b}<=>$surveydata{$a};
}
```

VisualC++で始めるWindowsプログラミング（3）
DIBを直接制御するには

菊地 功 Kikuchi Isawo

なんとなくグラフィックツール作成講座っぽくなっている本連載だが，今回はさらに進んで，WindowsのGDI任せにしていた描画処理を自前で処理することを考えてみよう。また，IEなどでお馴染みの「ReBar」も追加してみることにする。

この連載はVisual C++の入門記事だったはずなのに，ふと気づくとなんかかなり偏った方向に進んでいるような気がする。入門には違いなさそうだけど。でも今号の特集はグラフィックということなので，それに乗じて，もうちょっとだけ画像について触れてみたい。春号の最後でも述べたとおり，結局デバイスコンテキストなんかにお世話になっている限り，まっとうな画像処理なんてできはしない。そこで，画像編の締めとして，ビット列のダイレクトアクセスに挑戦してみよう。

その昔，X680x0でそういうことをやろうと思ったら，VRAMに直接書き込むのが当たり前であったことに異論を唱えるものはいまい。というより，それしかなかったといっても過言ではない（SX-Windowを除いて）。だが，Windowsではそうもいかない。OS自体がそういうことを許していないし，解像度も色数もVRAM容量もまちまち，仮にVRAMに直接アクセスできたとしても，ビデオカードによってメモリ構造が異なる可能性があるというのは以前に述べたとおりだ。DirectDrawを使えばそれらしいことができそうだが，それはまた別の話だ。

だが裏を返せば，なにか画像を表示しようと思ったら，そのビット列はメインメモリ上に存在しているということである。これは考えようによってはVRAMを直接いじるよりも好都合だ。なぜなら，画面サイズやVRAM容量に左右されることなく（メインメモリの許す限り）好きなサイズの画像を作れるし，しかもクリッピングなどは考える必要がないからだ。表示に関しては，結局デバイスコンテキストの世話になるが，それはWindowsの流儀であり，そうしないアプリケーションは（DirectDrawを除いて）反則であるので，おとなしく従っておけばよい。

ビットマップ配列はいずこ

ではそのビット列はどこに，どのように格納されているか。って，そんなに謎掛けるほどのもんじゃない。前回でも使ったDIBハンドルの先だ。正確には，DIBハンドルで識別されるグローバルメモリの中。前回は単にそのハンドルを右から左に流しただけだったが，今回はこのDIBメモリをちょっくら探索しようって話だ。でもって，DIBのデータ構造がわかれば，すべては思いのままである。

DIBの構造は，実はBMPファイルからファイルヘッダ部分を取り除いたものとまったく同じである。いずれにせよ，ファイルを読み込む時点でBMPのフォーマットが必要になるので，そちらも一緒に説明しよう。

BMPファイルは，BITMAPFILEHEADERとBITMAPINFO，それにビット列から構成される（表1）。このBITMAPFILEHEADERこそがBMPにあってDIBにない部分である。つまり，DIBとはBITMAPINFOとビット列をあわせたものである。

●BITMAPFILEHEADER構造体

WORD bfType;
ファイルの種類を示す。'MB'（リトルエンディアン）でなければならない。

DWORD bfSize;
ファイルのサイズがバイト単位で入る。

WORD bfReserved1;
予約。0でなければならない。

WORD bfReserved2;
同上。

DWORD bfOffBits;
ビット列までのオフセットを示す。つまり，BITMAPINFOEHADERとBITMAPINFOの合計バイト数である。

BITMAPINFOは，さらにヘッダ部分のBITMAPINFOHEADERと，パレット部分（あれば）のRGBQUAD配列に細分化されるが，BMPのバージョンによってBITMAPINFOの代わりにBITMAPCOREINFOやBITMAPLONGINFOが格納されている場合もある。BITMAPCOREINFOはWindows 2.x以前で，BITMAPLONGINFOはOS/2で使われていたフォーマットらしいが，一応読み込みはサポートしておいたほうがいいだろう。ちなみに，BITMAPLONGINFOとBITMAPLONGHEADER構造体はヘッダファイルとしては用意されていない。

●BITMAPINFOHEADER構造体

DWORD biSize;
この構造体のサイズ（sizeof（BITMPAINFOHEADER）=40）。

LONG biWidth;
画像の幅（ピクセル）。

LONG biHeight;
画像の高さ（ピクセル）。

WORD biPlanes;
プレーン数。通常は1。

WORD biBitCount;
ビットカウント（1，4，8，16，24，32）。

DWORD biCompression;
圧縮フラグ。BI_RGBが非圧縮，BI_RLE8が256色の，BI_RLE4が16色のRLE圧縮。

DWORD biSizeImage;
ビット列のサイズ。非圧縮のときは0でよい。

LONG biXPelsPerMeter;
横方向の解像度。あまり気にしなくてよい（たいていは0）。

LONG biYPelsPerMeter;
縦方向の解像度。同上。

DWORD biClrUsed;
パレッタイズBMPの場合の，使用しているパレット数（次に続くQGBQUAD配列の数）。0の場合は，そのビットカウントで利用できる最大数。

DWORD biClrImportant;
特に重要なパレットの数。減色の際などに使われるようだが，まじめに指定されていることは稀。0の場合は，すべてが重要。

BITMAPCOREHEADERとBITMAPLONGHEADERも，メンバのサイズが違ったりはするが，基本的に同じだ。BITMAPLONGHEADERのblUnits以降は無視していいだろう。

さらに，実際に見たことはないのだが，BITMAPINFOHEADERの代わりに，Windows 95からはBITMAPV4HEADERが，Windows 98からはBITMAPV5HEADERがサポートされたらしい（BITMAPV5HEADERのほうはヘルプ内に"This is preliminary documentation and subject

to change."という記載があるが)。ただし，これらはBITMAPINFOHEADERの後ろに情報が付加されるかたちで拡張されているので，BITMAPINFOHEADERと同じように情報を取得し，biSizeでパレットまでスキップさせるかたちにすれば特に問題はないはずだ。ただ，ちょっと気になったのは，Windows 98以降ではbiCompressionにBI_JPEGというフラグが増えたこと。いうまでもなくJPEG圧縮なわけだが，実際に機能するのかどうかも不明なため，今回は触れないことにする。RLE圧縮については後述する。

ちょっと注意が必要なのは，16ビットと32ビットカラーのBMPだ。以前のドキュメントにはビットカウントに16と32は含まれていなかったのだが，いつのまにか増えている。あんまり目にすることもないんじゃないかと思うが，一応サポートせねばなるまい。

16ビットの場合は，各WORD値に対して，下位から5ビットずつがBGRとなり，最上位ビットは使われない。32ビットの場合は，各DWORD値に対して，下位から8ビットずつがBGRとなり，上位8ビットは使われない。biCompressionがBI_BITFIELDSの場合はこの限りではないが，BI_BITFIELDSはシステムに依存してしまうので，流通はしないであろうと勝手に思い込むことにする。また，BITMAPV4HEADERやBITMAPV5HEADERの場合は，ヘッダ内にRGB(とα)のマスク情報が含まれており，本来ならこれを参照しなければならないのだろうが，このヘッダを持ったBMPも手元にないし，無視することにする。

BMPの特徴として注意が必要なことは，1ラインのバイト数がDWORD境界である(非圧縮の場合)ことと，スキャンラインが下から上に向かって格納されているということだ(biHeightにマイナス値を指定することで，上から下に格納もできるが，あまり一般的ではないだろう)。1ラインがDWORD境界ということは，たとえば，4ビットカラーで幅が4ドットの画像の場合，1ラインは2バイトになるはずだが，DWORD境界により余計な2バイトが付加され，1ラインは4バイトとなる。

さて，BMPといっても内部的にいろいろな構造があるので，いちいちそれぞれに対応した描画ルーチンを持たせるというのは面倒だ。そこで，ファイルのロード時点でまずBITMAPINFOHEADERを持った非圧縮24ビットカラーに変換してしまうことにする。それがいちばん手っ取り早くて簡単だろう。もちろん用途によってはパレットのまま編集したいということもあるだろうが，それは自分で挑戦してみてもらいたい。

■RLE圧縮

24ビットカラーへの変換などはとりわけ難しいことではないが，RLE圧縮のフォーマットについては知っておかないことにはどうしようもない。BMPのRLEは単純なランレングス圧縮であり，8ビットカラー時のものがBI_RLE8，4ビットカラー時のものがBI_RLE4である。つまり，それ以外のビットカウントではRLE圧縮はできない。

ご存じだと思うが，ランレングスとはひとつのデータとそれが連続する個数で記述する原始的な圧縮法である。したがって，同じ値の連続性が高いデータには有効だが，連続性が低い場合には，むしろデータが大きくなってしまう。極端に大きくなることを防ぐために，部分的にベタで記述するコー

表1 BMPファイルを構成する構造体

●ファイルヘッダ
```
typedef struct tagBITMAPFILEHEADER {
    WORD     bfType;
    DWORD    bfSize;
    WORD     bfReserved1;
    WORD     bfReserved2;
    DWORD    bfOffBits;
} BITMAPFILEHEADER;
```

●もっとも一般的に使われているインフォメーションブロック
```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER    bmiHeader;
    RGBQUAD             bmiColors[1];
} BITMAPINFO;

typedef struct tagBITMAPINFOHEADER{
    DWORD    biSize;
    LONG     biWidth;
    LONG     biHeight;
    WORD     biPlanes;
    WORD     biBitCount;
    DWORD    biCompression;
    DWORD    biSizeImage;
    LONG     biXPelsPerMeter;
    LONG     biYPelsPerMeter;
    DWORD    biClrUsed;
    DWORD    biClrImportant;
} BITMAPINFOHEADER;

typedef struct tagRGBQUAD {
    BYTE     rgbBlue;
    BYTE     rgbGreen;
    BYTE     rgbRed;
    BYTE     rgbReserved;
} RGBQUAD;
```

●Windows 2.x以前で使われていたインフォメーション
```
typedef struct _BITMAPCOREINFO {
    BITMAPCOREHEADER    bmciHeader;
    RGBTRIPLE           bmciColors[1];
} BITMAPCOREINFO;

typedef struct tagBITMAPCOREHEADER {
    DWORD    bcSize;
    WORD     bcWidth;
    WORD     bcHeight;
    WORD     bcPlanes;
    WORD     bcBitCount;
} BITMAPCOREHEADER;

typedef struct tagRGBTRIPLE {
    BYTE rgbtBlue;
    BYTE rgbtGreen;
    BYTE rgbtRed;
} RGBTRIPLE;
```

●OS/2で使われていたインフォメーション
```
typedef struct tagBITMAPLONGINFO {
    BITMAPLONGHEADER    bmliHeader;
    RGBQUAD             bmliColors[1];
} BITMAPLONGINFO;

typedef struct tagBITMAPLONGHEADER {
    DWORD    blSize;
    short    blWidth;
    short    blHeight;
    WORD     blPlanes;
    WORD     blBitCount;
    DWORD    blCompression;
    DWORD    blSizeImage;
    LONG     blXPelsPerMeter;
    LONG     blYPelsPerMeter;
    DWORD    blClrUsed;
    DWORD    blClrImportant;
    DWORD    blUnits;
    DWORD    blReserved;
    DWORD    blRendering;
    DWORD    blSize1;
    DWORD    blSize2;
    DWORD    blColorEncoding;
    DWORD    blIdentifier;
} BITMAPLONGHEADER;
```

●Windows 95以降でサポートされたらしいヘッダ
```
typedef struct {
    DWORD           bV4Size;
    LONG            bV4Width;
    LONG            bV4Height;
    WORD            bV4Planes;
    WORD            bV4BitCount;
    DWORD           bV4V4Compression;
    DWORD           bV4SizeImage;
    LONG            bV4XPelsPerMeter;
    LONG            bV4YPelsPerMeter;
    DWORD           bV4ClrUsed;
    DWORD           bV4ClrImportant;
    DWORD           bV4RedMask;
    DWORD           bV4GreenMask;
    DWORD           bV4BlueMask;
    DWORD           bV4AlphaMask;
    DWORD           bV4CSType;
    CIEXYZTRIPLE    bV4Endpoints;
    DWORD           bV4GammaRed;
    DWORD           bV4GammaGreen;
    DWORD           bV4GammaBlue;
} BITMAPV4HEADER;
```

●Windows 98以降でサポートされたらしいヘッダ
```
typedef struct {
    DWORD           bV5Size;
    LONG            bV5Width;
    LONG            bV5Height;
    WORD            bV5Planes;
    WORD            bV5BitCount;
    DWORD           bV5Compression;
    DWORD           bV5SizeImage;
    LONG            bV5XPelsPerMeter;
    LONG            bV5YPelsPerMeter;
    DWORD           bV5ClrUsed;
    DWORD           bV5ClrImportant;
    DWORD           bV5RedMask;
    DWORD           bV5GreenMask;
    DWORD           bV5BlueMask;
    DWORD           bV5AlphaMask;
    DWORD           bV5CSType;
    CIEXYZTRIPLE    bV5Endpoints;
    DWORD           bV5GammaRed;
    DWORD           bV5GammaGreen;
    DWORD           bV5GammaBlue;
    DWORD           bV5Intent;
    DWORD           bV5ProfileData;
    DWORD           bV5ProfileSize;
    DWORD           bV5Reserved;
} BITMAPV5HEADER;
```

リスト1　BMPファイルを24ビットDIBにする関数

```
HDIB LoadBMPFile(CFile &file )
{
  BITMAPFILEHEADER  bmfHeader;
  BITMAPINFOHEADER  bmiHeader;
  HDIB  hDIB;
  LPSTR  pDIB;
  WORD  BitCount;
  DWORD  Size, Compression, ClrUsed;
  int  i, x, y, shift, pal, PalBytes;
  unsigned char  *rgb, *buf;
  unsigned char  *dib;
  LONG  BMPWidthBytes, DIBWidthBytes;

  // ファイルヘッダを確認
  if((file.Read((LPSTR)&bmfHeader,sizeof(bmfHeader))! =
    sizeof(bmfHeader))||(bmfHeader.bfType! = 'MB') )
    return NULL;
  // ヘッダサイズの取得
  file.Read((LPSTR)&Size, sizeof(DWORD) );
  // BITMAPINFOHEADERに値を格納
  if(Size =  = sizeof(BITMAPCOREHEADER) ){// Windows 1.x,2.x OS/2 1.x
    BITMAPCOREHEADER bmcHeader;
    bmcHeader.bcSize = Size;
    file.Read(((LPSTR)&bmcHeader)+sizeof(DWORD), bmcHeader.bcSize-sizeof(DWORD) );
    bmiHeader.biSize = 40;
    bmiHeader.biWidth = bmcHeader.bcWidth;
    bmiHeader.biHeight = bmcHeader.bcHeight;
    bmiHeader.biPlanes = bmcHeader.bcPlanes;
    bmiHeader.biBitCount = bmcHeader.bcBitCount;
    bmiHeader.biCompression = BI_RGB;
    bmiHeader.biSizeImage = 0;
    bmiHeader.biXPelsPerMeter = 0;
    bmiHeader.biYPelsPerMeter = 0;
    bmiHeader.biClrUsed = 0;
    bmiHeader.biClrImportant = 0;
    PalBytes = sizeof(RGBTRIPLE);
  }else if(Size =  = sizeof(BITMAPLONGHEADER) ){ // OS/2 2.x
    BITMAPLONGHEADER bmlHeader;
    bmlHeader.blSize = Size;
    file.Read(((LPSTR)&bmlHeader)+sizeof(DWORD), bmlHeader.blSize-sizeof(DWORD) );
    bmiHeader.biSize = 40;
    bmiHeader.biWidth = bmlHeader.blWidth;
    bmiHeader.biHeight = bmlHeader.blHeight;
    bmiHeader.biPlanes = bmlHeader.blPlanes;
    bmiHeader.biBitCount = bmlHeader.blBitCount;
    bmiHeader.biCompression = bmlHeader.blCompression;
    bmiHeader.biSizeImage = bmlHeader.blSizeImage;
    bmiHeader.biXPelsPerMeter = bmlHeader.blXPelsPerMeter;
    bmiHeader.biYPelsPerMeter = bmlHeader.blYPelsPerMeter;
    bmiHeader.biClrUsed = bmlHeader.blClrUsed;
    bmiHeader.biClrImportant = bmlHeader.blClrImportant;
    PalBytes = sizeof(RGBQUAD);
  }else { // Windows 3.x or later
    bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
    file.Read(((LPSTR)&bmiHeader)+sizeof(DWORD), bmiHeader.biSize-sizeof(DWORD) );
    file.Seek(Size-bmiHeader.biSize, CFile::current );
    PalBytes = sizeof(RGBQUAD);
  }
  BitCount = bmiHeader.biBitCount;
  Compression = bmiHeader.biCompression;
  ClrUsed = bmiHeader.biClrUsed;
  // 24ビット非圧縮に設定
  bmiHeader.biBitCount = 24;
  bmiHeader.biCompression = BI_RGB;
  bmiHeader.biClrUsed = 0;
  // BMPファイルの1ラインのバイト数
  BMPWidthBytes = (bmiHeader.biWidth*BitCount+7)/8;
  BMPWidthBytes = (BMPWidthBytes+3)/4*4;  // 4バイト境界
  // 24ビットDIBの1ラインのバイト数
  DIBWidthBytes = bmiHeader.biWidth*sizeof(RGBTRIPLE);
  DIBWidthBytes = (DIBWidthBytes+3)/4*4;  //4バイト境界
  // DIBのサイズ
  Size = (DWORD)DIBWidthBytes*(DWORD)bmiHeader.biHeight
    +sizeof(BITMAPINFOHEADER);
  // DIBを確保
  hDIB = (HDIB)::GlobalAlloc(GMEM_MOVEABLE|GMEM_ZEROINIT, Size );
  if(hDIB =  = NULL)return NULL;
  // DIBをロック
  pDIB = (LPSTR)::GlobalLock((HGLOBAL)hDIB );
  // ヘッダを書きこみ
  *((BITMAPINFOHEADER far *)pDIB) = bmiHeader;
  // パレット数を取得
  if(ClrUsed =  = 0 ){
    switch(BitCount ){
    case 1:  ClrUsed = 2;
        break;
    case 4:  ClrUsed = 16;
        break;
    case 8:  ClrUsed = 256;
        break;
    }
  }
  // パレットを取得
  if(BitCount< = 8 ){
    rgb = new unsigned char [ClrUsed*sizeof(RGBTRIPLE)];
    if(PalBytes =  = sizeof(RGBTRIPLE) ){
      file.Read((LPSTR)rgb, (int)ClrUsed*sizeof(RGBTRIPLE) );
    }else {
      for(i = 0; i<(int)ClrUsed; i++ ){
        file.Read((LPSTR)&rgb[i*sizeof(RGBTRIPLE)], sizeof(RGBTRIPLE) );
        file.Seek(sizeof(RGBQUAD)-sizeof(RGBTRIPLE), file.current );
      }
    }
  }
  dib = (unsigned char *)((DWORD)pDIB+sizeof(BITMAPINFOHEADER));
  if(BitCount =  = 24 ){ // 最初から24ビットならそのままリード
    Size = (DWORD)DIBWidthBytes*(DWORD)bmiHeader.biHeight;
    if(file.ReadHuge(dib, Size )! = Size ){
      ::GlobalUnlock((HGLOBAL) hDIB);
      ::GlobalFree((HGLOBAL) hDIB);
      return NULL;
    }
  }else if(Compression =  = BI_RGB ){
    buf = new unsigned char [BMPWidthBytes];
    switch(BitCount ){
    case 1:  for(y = 0; y<bmiHeader.biHeight; y++ ){
          // 1ライン分読みこむ
          if(file.Read((LPSTR)buf, (UINT)BMPWidthBytes )! = (unsigned)BMPWidthBytes
){
            delete buf; delete rgb;
            ::GlobalUnlock((HGLOBAL) hDIB);
            ::GlobalFree((HGLOBAL) hDIB);
            return NULL;
          }
          for(x = i = 0, shift = 7; x<bmiHeader.biWidth; x++, shift-- ){
            if(shift<0 ){ i++; shift = 7; }
            pal = (buf[i]>>shift)&1;
            memcpy(dib, &rgb[pal*sizeof(RGBTRIPLE)], 3 );
            dib + =  3;
          }
          // 改ライン(DWORD境界の詰め物をスキップ)
          dib + =  DIBWidthBytes-bmiHeader.biWidth*sizeof(RGBTRIPLE);
        }
        break;
    case 4:  for(y = 0; y<bmiHeader.biHeight; y++ ){
          if(file.Read((LPSTR)buf, (UINT)BMPWidthBytes )! = (unsigned)BMPWidthBytes
){
            delete buf; delete rgb;
            ::GlobalUnlock((HGLOBAL) hDIB);
            ::GlobalFree((HGLOBAL) hDIB);
            return NULL;
          }
          for(x = i = 0, shift = 4; x<bmiHeader.biWidth; x++, shift- = 4 ){
            if(shift<0 ){ i++; shift = 4; }
            pal = (buf[i]>>shift)&0xf;
            memcpy(dib, &rgb[pal*sizeof(RGBTRIPLE)], 3 );
            dib + =  3;
          }
          dib + =  DIBWidthBytes-bmiHeader.biWidth*sizeof(RGBTRIPLE);
        }
        break;
    case 8:  for(y = 0; y<bmiHeader.biHeight; y++ ){
          if(file.Read((LPSTR)buf, (UINT)BMPWidthBytes )! = (unsigned)BMPWidthBytes
){
            delete buf; delete rgb;
            ::GlobalUnlock((HGLOBAL) hDIB);
            ::GlobalFree((HGLOBAL) hDIB);
            return NULL;
          }
          for(x = 0; x<bmiHeader.biWidth; x++ ){
            pal = buf[x];
            memcpy(dib, &rgb[pal*sizeof(RGBTRIPLE)], 3 );
            dib + =  3;
          }
          dib + =  DIBWidthBytes-bmiHeader.biWidth*sizeof(RGBTRIPLE);
        }
        break;
    case 16:
      {
        WORD *pixel;
        for(y = 0; y<bmiHeader.biHeight; y++ ){
          if(file.Read((LPSTR)buf, (UINT)BMPWidthBytes )! = (unsigned)BMPWidthBytes
){
            delete buf;
            ::GlobalUnlock((HGLOBAL) hDIB);
            ::GlobalFree((HGLOBAL) hDIB);
            return NULL;
          }
          pixel = (WORD*)buf;
          for(x = 0; x<bmiHeader.biWidth; x++ ){
            *(dib++) = ((*pixel)<<3)&0xf8;
            *(dib++) = ((*pixel)>>2)&0xf8;
            *(dib++) = ((*pixel)>>7)&0xf8;
            pixel++;
          }
          dib + =  DIBWidthBytes-bmiHeader.biWidth*sizeof(RGBTRIPLE);
        }
        break;
```

```
    }
  case 32:
    {
      DWORD *pixel;
      for(y = 0; y<bmiHeader.biHeight; y++ ){
        if(file.Read((LPSTR)buf, (UINT)BMPWidthBytes )! = (unsigned)BMPWidthBytes
){
          delete buf;
          ::GlobalUnlock((HGLOBAL) hDIB);
          ::GlobalFree((HGLOBAL) hDIB);
          return NULL;
        }
        pixel = (DWORD*)buf;
        for(x = 0; x<bmiHeader.biWidth; x++ ){
          *(dib++) = (*pixel)&0xff;
          *(dib++) = ((*pixel)>>8)&0xff;
          *(dib++) = ((*pixel)>>16)&0xff;
          pixel++;
        }
        dib + =  DIBWidthBytes-bmiHeader.biWidth*sizeof(RGBTRIPLE);
      }
      break;
    }
  }
  delete buf;
  if(BitCount< = 8)delete rgb;
}else { // RLE
   :
  省略
   :
}
::GlobalUnlock((HGLOBAL) hDIB);
return hDIB;
}
```

ドを持っているのが普通だ。画像の場合では，アニメ調の絵には強いが，自然画にはめっぽう弱いということになる。

BMPの場合はというと，エンコードモードと絶対モードを持っている。たいそうな名前だが，要するにエンコードモードはランレングス部分，絶対モードはベタの部分である(以下はBI_RLE8の場合)。

●エンコードモード

2バイト単位で構成される。先頭の1バイトは個数を示し，2バイト目はカラーインデックスを示す。つまり，最大255バイトを2バイトに圧縮できる。もし1バイト目が0の場合，2バイト目は次の意味を持つ。

0 ラインの終端
1 ビットマップの終端
2 偏移を示す。後ろに続く2バイトは，それぞれ次のピクセルへの水平・垂直オフセットとなる(そこまでスキップする)

●絶対モード

第1バイトに0，第2バイトにはその後ろにベタで記述されるカラーインデックスの個数を示す。2個以下の場合は結果的にエンコードモードで記述してもバイト数が絶対モード以下になるため，3以上の値となる。また，ワード境界となるように0が付加される。

つまり，次がどちらのモードなのかは，第1バイトが0かつ第2バイトが3以上の場合は絶対モード，それ以外はエンコードモードということになる。BI_RLE4でも基本的に同じだが，4ビットのカラーインデックスは1バイトに2個パックされる。つまり，エンコードモードの場合は，2ドットをセットとして個数を指定できるので，ディザリングも圧縮できる。

そんなわけで，BMPファイルを読み込み，24ビット非圧縮のDIBを構築する関数がリスト1だ。ちょっと長くなってしまったので，RLE展開部分はここでは省略してあるが，付録CD-ROMの中のリストはもちろん対応している。例によってコメントは少なく，昔作った関数の焼き直しなのであまり美しくはないが，面倒なだけで難しいことはしていない。そうそう，今回はもうパレットは無視してしまったので，ハイカラー以上の環境で見ないと悲しいことになる。

古のラインルーチン

DIBが取れてしまえば，表示は前回と同じだ。ただ，今回は表示することが目的じゃない。ビットマップに直接書き込むことが目的だ。わかっていると思うが，デバイスコンテキストを使っていたときは，ほいっと座標を渡してやるだけで，びゃーっとラインを引いてくれたりしていたが，そーゆーことも自分でやらなきゃいけない。

というわけで，基本はやっぱりラインルーチンなのである。なにをいまさらという人は，この先をちょっと読み飛ばしてもらって構わない。

まあ確かに方程式立ててまじめに計算するとか，浮動小数点演算するとか，いまどきのCPUならそれで線の100本や200本引いてもびくともしないんだろうが(そうだろうか？)，そこはそれ。やっぱり少しでも速いほうがいいに決まってるし，実は古のテクのほうが理屈も簡単なんである。

いまAからBへ直線を引くとしよう(図1)。ここで，x座標がx1からx2へ進む途中，何ドット進むとy方向へ1ドット進めればよいかを考える。xが1ドット進むと，yは理想の座標よりもdy/dxだけずれることがわかるだろう。もう1ドット進むと，さらに誤差は倍になって，2＊dy/dxとなる。もしxがnドット目で誤差n＊dy/dxが四捨五入して1(つまり0.5)以上になったとき，yは1ドット進むことになり，その後の誤差はn＊dy/dx-1=(n＊dy-dx)/dxとなる。

つまり，誤差のカウンタを用意して，xが1ドット進むたびにdy/dxを加え，もしそれが四捨五入して1以上になったらyの座標を加算し，カウンタから1を引く，それを繰り返せばよいことがおわかりだろう。ただ，dy/dxという小数を扱うのはうれしくない。そこで，全体をdx倍してみると，「xが1ドット進むたびにdyを加え，もしそれがdx以上になったらyの座標を加算し，カウンタからdxを引く」となる。なんてシンプルなんだろう。ざっとプログラムを書いてみると，

```
int dx = x2-x1;
int dy = y2-y1;
int counter = dx/2;
int y = y1;
for( int x=x1; x<=x2; x++ ){
  pset( x, y );
  counter += dy;
  if( counter>=dx ){
    y++;
    counter -= dx;
  }
}
```

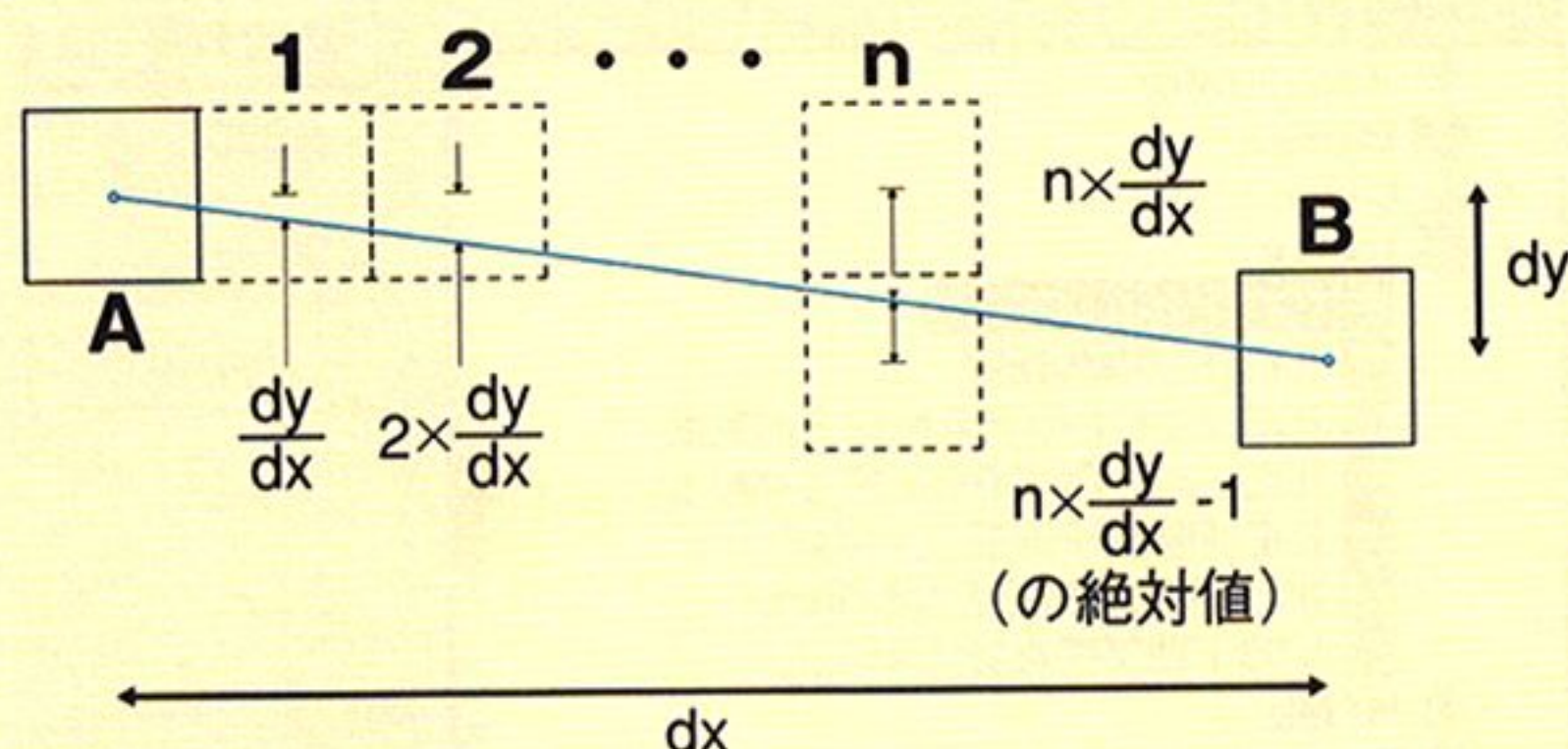



図1　ラインルーチンの概念

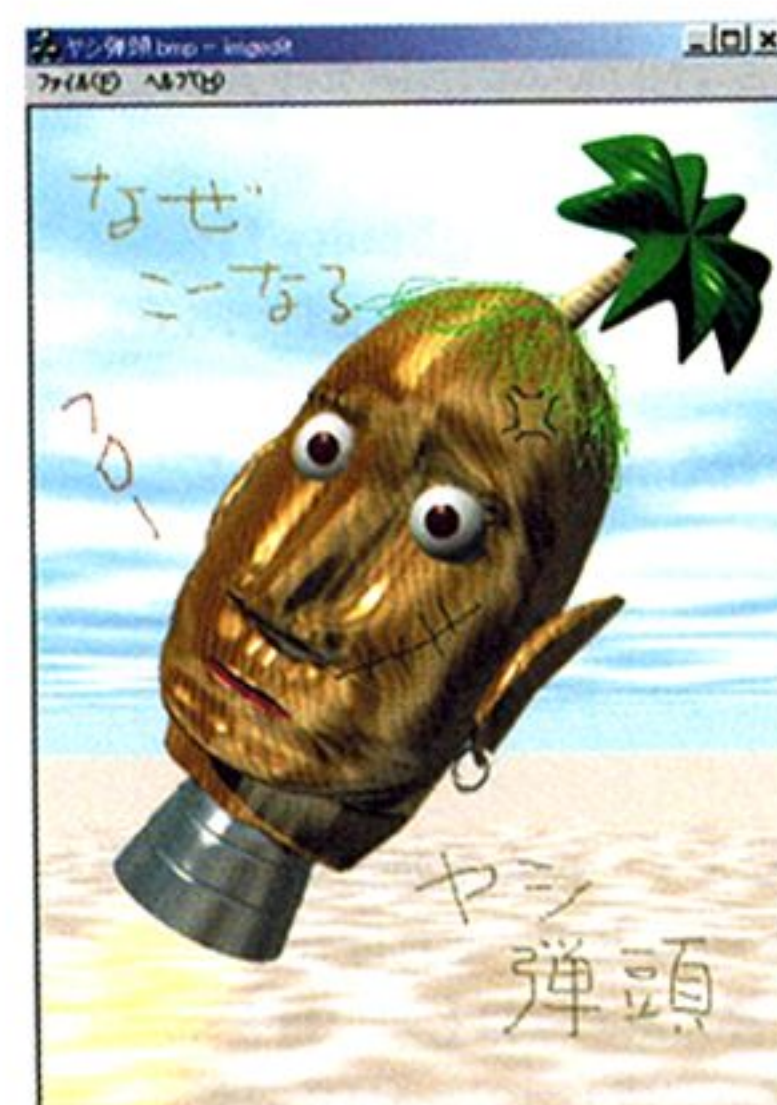



図2　サンプルImgEdit

pset()というのは点を打つ仮想命令だと思ってもらいたい。最初にcounterをdx/2で初期化しているのは，四捨五入の代わりである。たった，たったこれだけでラインが描けてしまうのだ。ハラショー！

だがこれには大きな落とし穴がある。いまはxを基準にy座標を追ったが，これが有効なのは，直線が45度よりもなだらかな場合に限られる。それよりも急な場合は，yを基準としなければならない。また，直線の方向によって，正負も入れ替えなければならない。そこで，もうちょっと改良する。

```
int dx = abs(x2-x1) ;
int dy = abs(y2-y1) ;
if( dx>dy ){
  if( x1>x2 ){ swap(x1,x2); swap(y1,y2); }
  int sign = (y1==y2)?0:((y1<y2)?1:-1);
  int counter = dx/2;
  int y = y1;
  for( int x=x1; x<=x2; x++ ){
    pset( x, y );
    counter += dy;
    if( counter>=dx ){
      y += sign;
      counter -= dx;
    }
  }
} else {
  if( y1>y2 ){ swap(x1,x2); swap(y1,y2); }
  int sign = (x1==x2)?0:((x1<x2)?1:-1);
  int counter = dy/2;
  int x = x1;
  for( int y=y1; y<=y2; y++ ){
    pset( x, y );
    counter += dx;
    if( counter>=dy ){
      x += sign;
      counter -= dy;
    }
  }
}
```

直線の傾きに応じて，xを基準とする場合とyを基準にする場合に分ける。さらにその基準の軸で，増加の向きになるようにポイントを入れ替える。で，もう一方の軸の増減値をsignで設定してやればよい。あと，直線に対称性を持たせるために，双方のポイントから互いに向かって点を算出していくという手法もあるが，今回はそこまではやっていない。

さて，問題はpset()部分である。これをDIBに対応させなければならないが，もうなんてことはないだろう。

```
void pset( int x, int y, RGBTRIPLE rgb )
{
  BITMAPINFOHEADER *pbmih = (BITMAPINFOHEADER*)
  GlobalLock( hDIB ) ;
  unsigned char *dib = (unsigned char *)(pbmih+1);
  int WidthBytes = (pbmih->biWidth*sizeof(RGBTRIPLE)
  +3)/4*4;
  dib += x*sizeof(RGBTRIPLE)+(pbmih->biHeight-y)
  *WidthBytes;
  *((RGBTRIPLE*)dib) = rgb;
  GlobalUnlock( hDIB );
}
```

「1ドット打つたびにアドレスを計算するなんて，なんて非効率的なことしてるんだ！」と思った君。君は正しい。ということで，それは君の宿題にしよう。次号までに必ずやってくるように。答えあわせはしないけど。

そんなこといってる間に，今回のサンプル完成(図2)。MDIである必要もないので，今回はSDIにしたが，表示部分やセーブの関数とかは前回のソースを引っ張ってきた。適当にBMPファイルを開いたら，マウスでぐりぐり落書きできる。この辺のことは，もう説明は不要だろう。

これだけでは寂しいので，右クリックで色を拾えるようにしておいた。やっていることは，右クリックした座標に対応したDIBのアドレスから，カラーを拾って保存しているだけだ。カレントカラーを表示するボックスなどはつけていないが，描き込んでみれば色が変わっていることがわかるだろう。なお，ファイルは保存すると無条件に24ビットカラーとなる。

IEっぽくしてみよう

実は今回はこれで終わりにしようかと思ったのだが，自分で読み返してみてふと気づいた。これじゃVisual C++の入門じゃなくて，画像操作の入門だよ。じゃあ，なにしようか。むー，としばらく考えて，ReBarをつけることにしてみた。ReBarってのはあれだ，IEとかにツールバーの代わりについてるやつで，左端のつまみ(グリップバー)をつかむとずりずりとスライドしてレイアウトできるバー(図3)。気持ち悪いって人もいるかもしれないけど，やっぱ使いようによってはカッコいい。これを載っけてみることにする。ところで，ReBarってなんて発音すればいいんだろう？　リバー？　レバー？

ところでこのReBar，単体では役に立たない。そりゃ，ダイアログだって中身のコントロールがなきゃ役に立たないけど，そうじゃなくて，ツールバーみたいに作ればすぐ使えるって代物じゃないってこと。というのは，このReBarは外側だけを制御するものであって，中身は別に作らなくちゃいけないからだ。もうちょっと専門的にカッコよくいうと，コンテナなんである。

では中身はどうするかというと，ツールバーとか，コントロールを別に作って，ReBarに載っけてやらなきゃいけない。しかも載っけられるコントロールはひとつのバー(ReBarの場合はバンドという)にひとつだけ。だから，複数のコントロールを載せたい場合には，まずダイアログ(チャイルドスタイルにすると，タイトルバーとかフレームはなくなる)とかに載せてやる必要がある。その代わり，ReBarひとつでバンドをいくつでも管理できてしまうので，インスタンスはひとつだけでよいのだ。

MFCには，ReBar作成用のクラスとして，CReBarとCReBarCtrlという2つのクラスが用意されている。かなり紛らわしいのだが，CReBarのほ

図3　Internet ExplorerのReBar

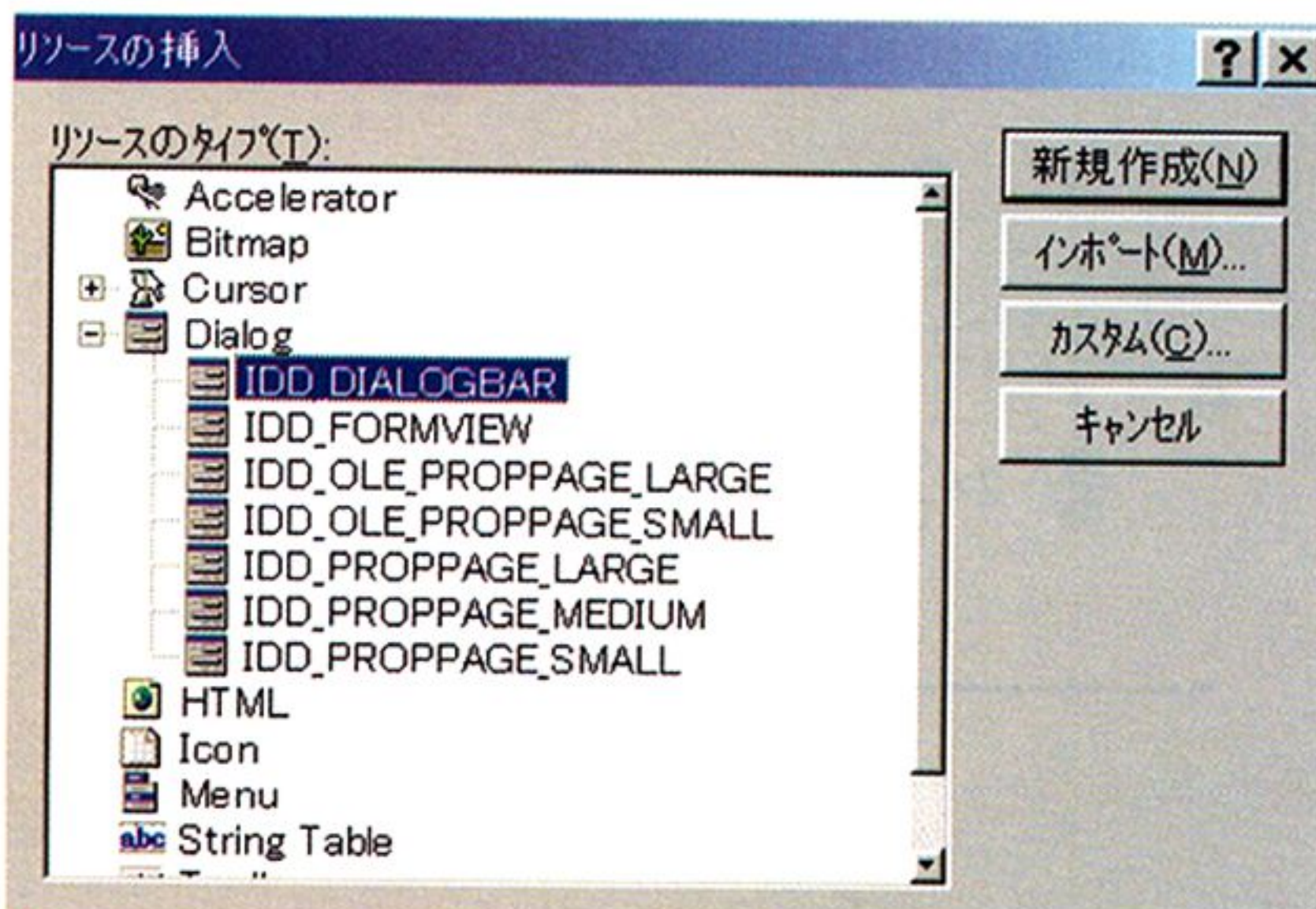



図4　IDD_DIALOGBARを選択すると，チャイルドウィンドウのダイアログフレームが作成される

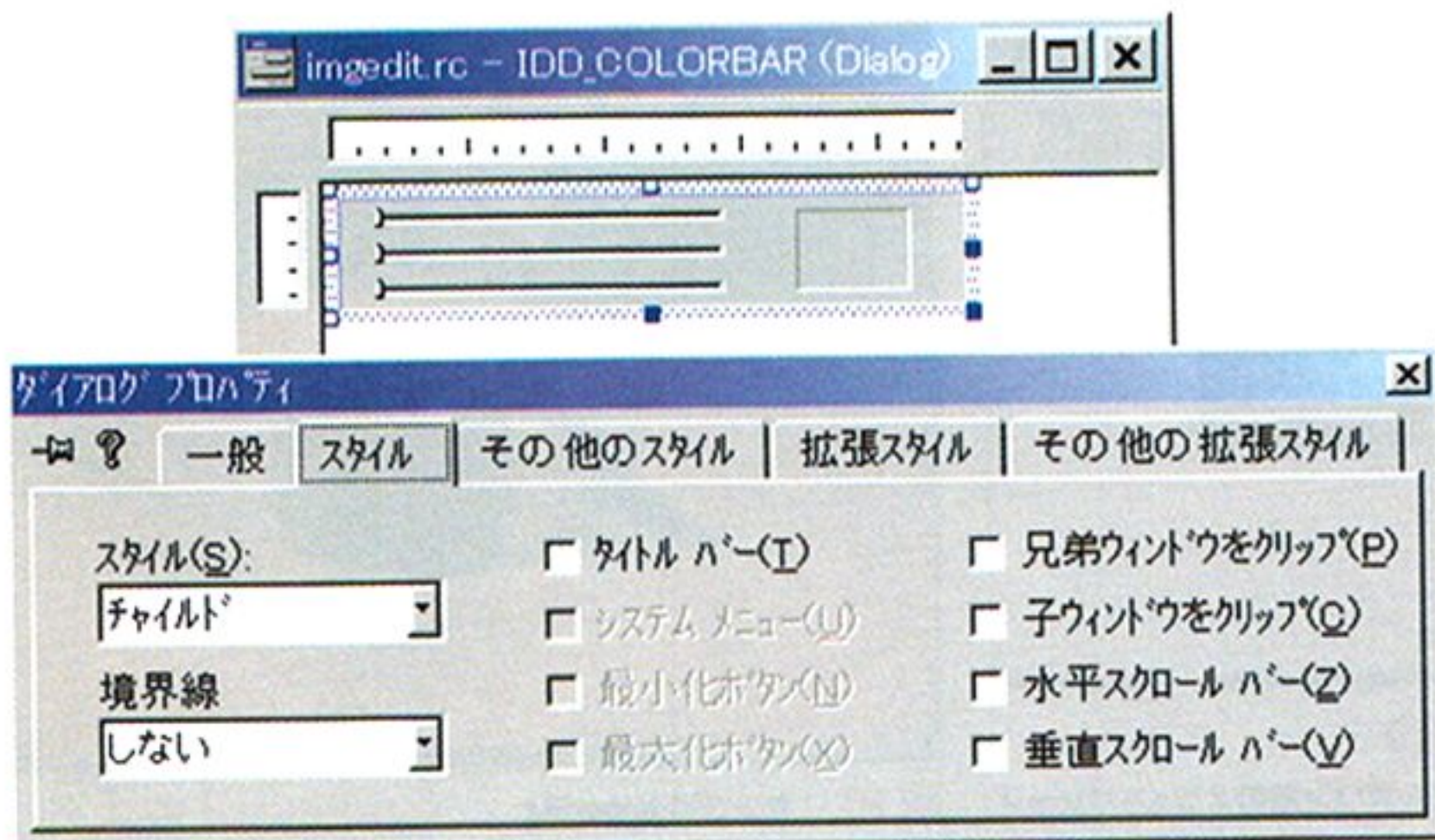



図5　ReBarに載せるダイアログのデザイン

図6　ダイアログクラスの作成

うが高水準で，設定や構造などを自動的に処理してくれるらしい。また，CReBarからCReBarCtrlのインスタンスを取得できるので，そういったものを手動で設定したい場合には，そちらのインスタンスを適宜取得して操作すればいいようだ。

ReBarに載っけるコントロールだが，ツールバーを1個と，ダイアログを1個載せてみよう。つまり，バンドは2つだ。ツールバーには，メニューの項目と，それに拡大表示ボタンもつけてみる。これにはプルダウンリストをつける。ボタンの右側に三角形のマークの縦長のボタンがついていて，それを押すとメニューがべろんと出てくるやつだ。

ダイアログのほうには，前回もやったRGBのスライダーを載せて，カレントカラーを設定できるようにしよう。ということで，まずはダイアログのフォームをデザインする。このとき，リソースの挿入で，IDD_DIALOGBARを選んでおけば，最初からタイトルバーもフレームもないチャイルドウィンドウ用のダイアログフレームを作ってくれる（図4）。

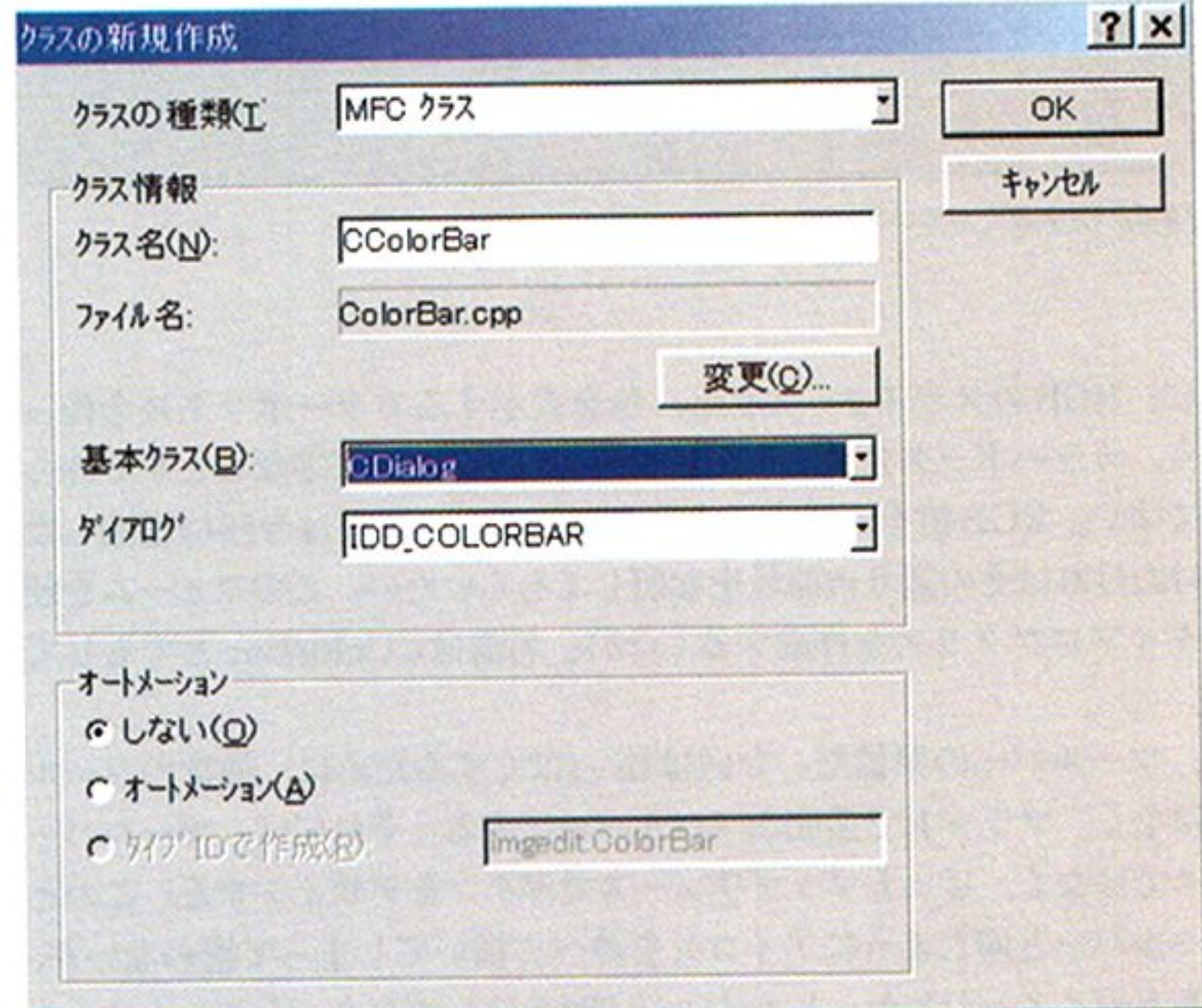
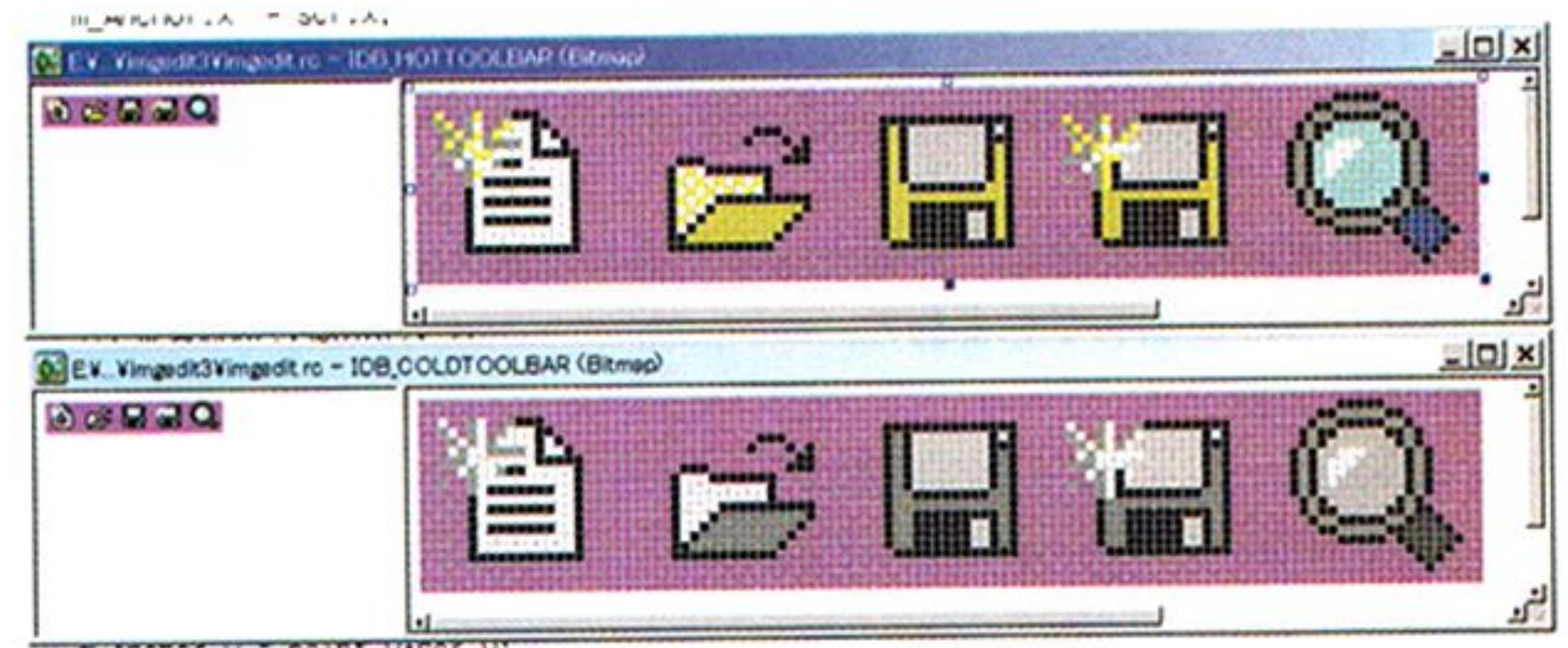



図7　ツールバー用のホットアイコンとデフォルトアイコン

リスト2　ReBarの設定

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
|
  if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
    return -1;

  // TODO: この位置に固有の作成用コードを追加してください

  // ReBarの作成
  if( !m_wndReBar.Create(this) )|
    TRACE0("Failed to create rebar\n");
    return -1;
  }
  // ToolBarの作成
  if( !m_wndToolBar.CreateEx(this) )|
    TRACE0("Failed to create toolbar\n");
    return -1;
  |
  // ToolBarの設定
  // ドロップダウンボタンを含む
  m_wndToolBar.GetToolBarCtrl().SetExtendedStyle( TBSTYLE_EX_DRAWDDARROWS );
  // ボタンイメージのセット
  CImageList img;
  // ホットボタンイメージ(紫が透明部分)
  img.Create( IDB_HOTTOOLBAR, 22, 0, RGB(255,0,255) );
  m_wndToolBar.GetToolBarCtrl().SetHotImageList(&img);
  img.Detach();
  // デフォルトボタンイメージ
  img.Create(IDB_COLDTOOLBAR, 22, 0, RGB(255, 0, 255));
  m_wndToolBar.GetToolBarCtrl().SetImageList(&img);
  img.Detach();
  // フラットな透明スタイル
  m_wndToolBar.ModifyStyle( 0, TBSTYLE_FLAT|TBSTYLE_TRANSPARENT );

  // ToolBarのボタン情報をセット
  CString str;
  // ボタンのIDを設定
  m_wndToolBar.SetButtonInfo( 0, ID_FILE_NEW, TBSTYLE_BUTTON, 0 );
  str.LoadString( ID_FILE_NEW );
  m_wndToolBar.SetButtonText( 0, str );  // ボタンにつくラベル
  m_wndToolBar.SetButtonInfo( 1, ID_FILE_OPEN, TBSTYLE_BUTTON, 1 );
  str.LoadString( ID_FILE_OPEN );
  m_wndToolBar.SetButtonText( 1, str );
  m_wndToolBar.SetButtonInfo( 2, ID_FILE_SAVE, TBSTYLE_BUTTON, 2 );
  str.LoadString( ID_FILE_SAVE );
  m_wndToolBar.SetButtonText( 2, str );
  m_wndToolBar.SetButtonInfo( 3, ID_FILE_SAVE_AS, TBSTYLE_BUTTON, 3 );
  str.LoadString( ID_FILE_SAVE_AS );
  m_wndToolBar.SetButtonText( 3, str );
  m_wndToolBar.SetButtonInfo( 4, 0, TBBS_SEPARATOR, 6 );
  // スケールボタンはドロップダウンリスト
        m_wndToolBar.SetButtonInfo(             5,          ID_SCALE_DROPDOWN,
TBSTYLE_BUTTON|TBSTYLE_DROPDOWN, 4 );
  str.LoadString( IDS_SCALE );
  m_wndToolBar.SetButtonText( 5, str );

  // カラーダイアログの作成
  if( !m_wndColor.Create( IDD_COLORBAR, this ) )|
    TRACE0("Failed to create colorbar\n");
    return -1;
  |

  // ToolBarとダイアログをReBarに追加
  m_wndReBar.AddBar( &m_wndToolBar );
  m_wndReBar.AddBar( &m_wndColor );

  // ReBarバンドのサイズと背景設定
  REBARBANDINFO rbbi;
  // ToolBar
  CRect rectBar;
  m_wndToolBar.GetItemRect( 0, &rectBar );
  rbbi.cbSize = sizeof(rbbi);
  rbbi.fMask = RBBIM_CHILDSIZE|RBBIM_IDEALSIZE|RBBIM_SIZE|RBBIM_BACKGROUND;
  rbbi.cxMinChild = rectBar.Width();
  rbbi.cyMinChild = rectBar.Height();
  rbbi.cx = rbbi.cxIdeal = rectBar.Width()*6;
  rbbi.hbmBack = LoadBitmap( AfxGetInstanceHandle(), MAKEINTRESOURCE(IDB_BARBK) );
  m_wndReBar.GetReBarCtrl().SetBandInfo( 0, &rbbi );

  // ColorBar
  rbbi.fMask = RBBIM_CHILDSIZE;
  m_wndReBar.GetReBarCtrl().GetBandInfo( 1, &rbbi );
  rbbi.fMask = RBBIM_IDEALSIZE|RBBIM_SIZE;
  rbbi.cx = rbbi.cxIdeal = rbbi.cxMinChild;
  m_wndReBar.GetReBarCtrl().SetBandInfo( 1, &rbbi );

  m_wndToolBar.SetBarStyle( m_wndToolBar.GetBarStyle()|
    CBRS_TOOLTIPS|CBRS_FLYBY|CBRS_SIZE_FIXED );

  return 0;
|
```

図8　2本のバンドがくっついている

これに，RGBのスライダー3本と，色を表示するカラーボックスを作っておこう。カラーボックスは，スタティックテキストの文字なしとかで作っておいてよい。RGB値を表示するテキストボックスなどは今回は省略したが，つけたければその辺りは前号を参照してもらいたい。このフォームを使って，ダイアログクラスを作成する（図6）。名前はCColorBarとでもしておこう。

次に，ツールバーの準備だ。今回はIEっぽくするために，通常のツールバーではなく，フラットで透明なツールバーにする。それには，ツールバーリソースではなく，ビットマップリソースでボタンをデザインする。このとき，ツールバーと同じようにアイコンを並べて描いてしまって構わないが，デフォルトのイメージとホットイメージは別々にしておく。ホットイメージというのは，ボタンの上にカーソルがきたときに表示されるイメージだ。

ホットイメージをカラーで描いておいて，デフォルトはそれをグレースケールにするのが一般的だろう。また，透明なボタンにするため，背景の部分は違う色にしておく。別に何色でも構わないのだが，やはりカラーキーといえば紫かな。

これで準備は整ったので，必要となるクラスのインスタンスを，CMainFrameクラスに追加してしまおう。

```
protected:
  CReBar       m_wndReBar;
  CToolBar     m_wndToolBar;
public:
  CColorBar    m_wndColor;
```

CColorBarだけpublicになっているのは，右クリックでカラーを拾ったときに，直接色を設定しに行けるようにだ。よくないことだとはわかっているのだが，遠回りするのも無駄だし面倒臭いもんで。

そうしたら，CMainFrameでWM_CREATEメッセージをハンドルし，リスト2のように追加する。コメントをざっと見れば，だいたいわかるだろう。ツールバーはフラットな透明ボタンにしているだけ，ダイアログのほうは普通にモードレスにするのと同じ要領だ。そうしておいて，AddBar()でReBarにバンドを登録している。

ちなみに，ボタンの表面につくラベルはストリングリソースから取得しており，ID_FILE_NEWなどは"新規"などの短い文字列にしてある。その後ろのREBARBANDINFO構造体だが，cxMinChildおよびcyMinChildにはボタン1個の大きさが入る。なんに使われるかというと，cyMinChildはバンドの高さ，cxMinChildのほうは，ほかのバンドによって重ねられない最小の幅となる。cxはバンドの初期の幅，cxIdealはグリップバーをクリックしたときに自動的に調節される幅になる。

ここまでで，すでにバンドは表示されるようになっているはずだ（図8）。実際，これでReBar自体の操作は完了なのである。グリップバーをつかんでバンドを入れ替えたり，クリックしてサイズの自動調整を試すとよいだろう。［新規］から［新規保存］までのボタンは，メニューのほうですでにインプリメントしてあるので，この時点で機能するようになっている。

残るはスケールのプルダウンメニューとその機能，カラーバー部分だ。カラーバーのスライダについては，前回にやったので，今回は特に説明はしない。スタティックテキストで作ったカラーボックスは，CColorBarクラス内でWM_CTLCOLORSTATICをハンドルすることで内部の色を変えられる。ON_MESSAGEマクロでメッセージ関数を作ったら，カレントカラーのソリッドブラシのハンドルを返すだけでよい。

さて，スケールのプルダウンメニューだが，実はツールバーは下向き三角のボタンの表示まではやってくれるが，それ以降はメッセージを発するだけでこちらの仕事となる。下向きボタンが押されると，ツールバー（デフォルトのIDはAFX_IDW_TOOLBAR）はTBN_DROPDOWNというNotifyメッセージを発するので，それをON_NOTIFYマクロで，

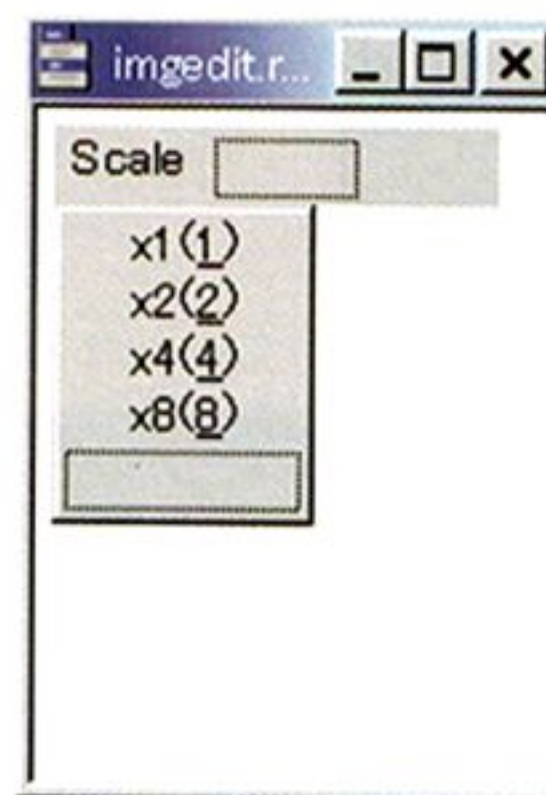



図9
プルダウン用のメニューリソース

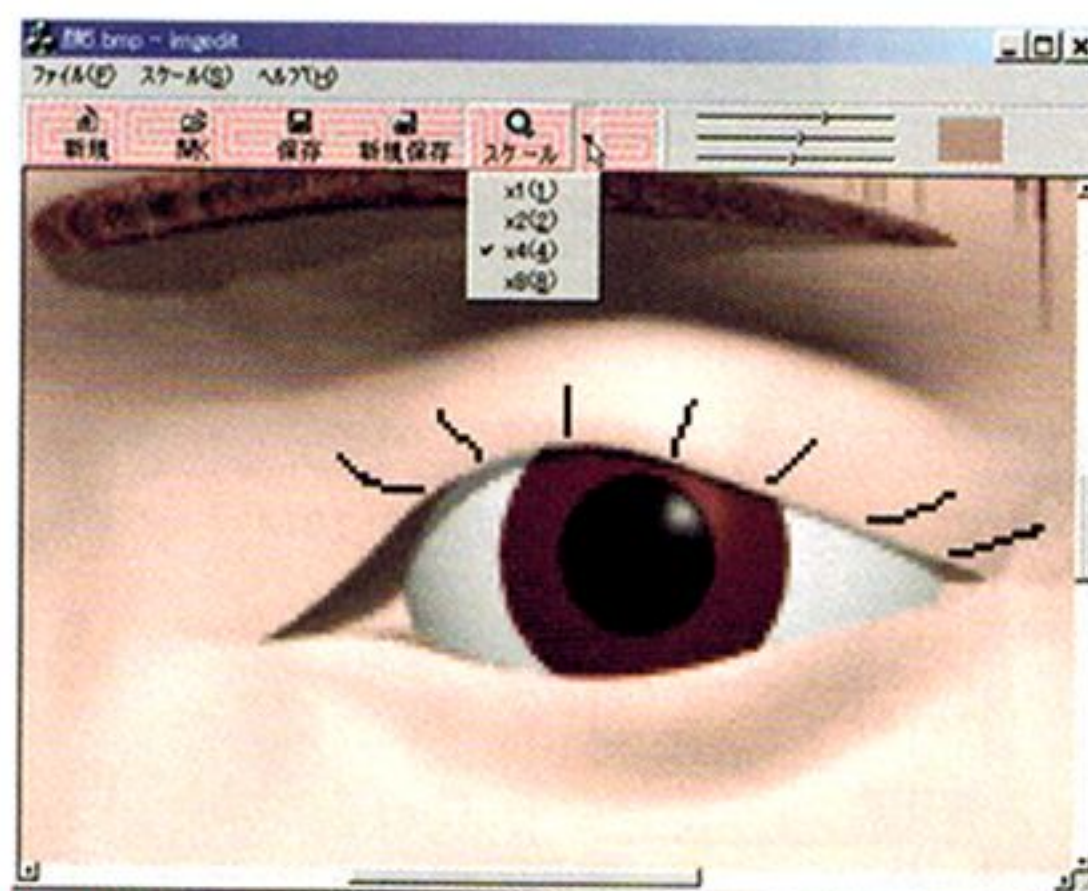



図10　サンプルimgedit2

```
ON_NOTIFY(TBN_DROPDOWN, AFX_IDW_TOOLBAR,
OnDropDown)
```

のようにハンドルし，メニューを表示してやる（リスト3）。

今回は，メニューはあらかじめ図9のようにリソースで作っておき，それをロードすることにした。ちょっとごちゃごちゃしているのは，メニューをちゃんと［スケール］ボタンの真下につながっているように表示するためだ。

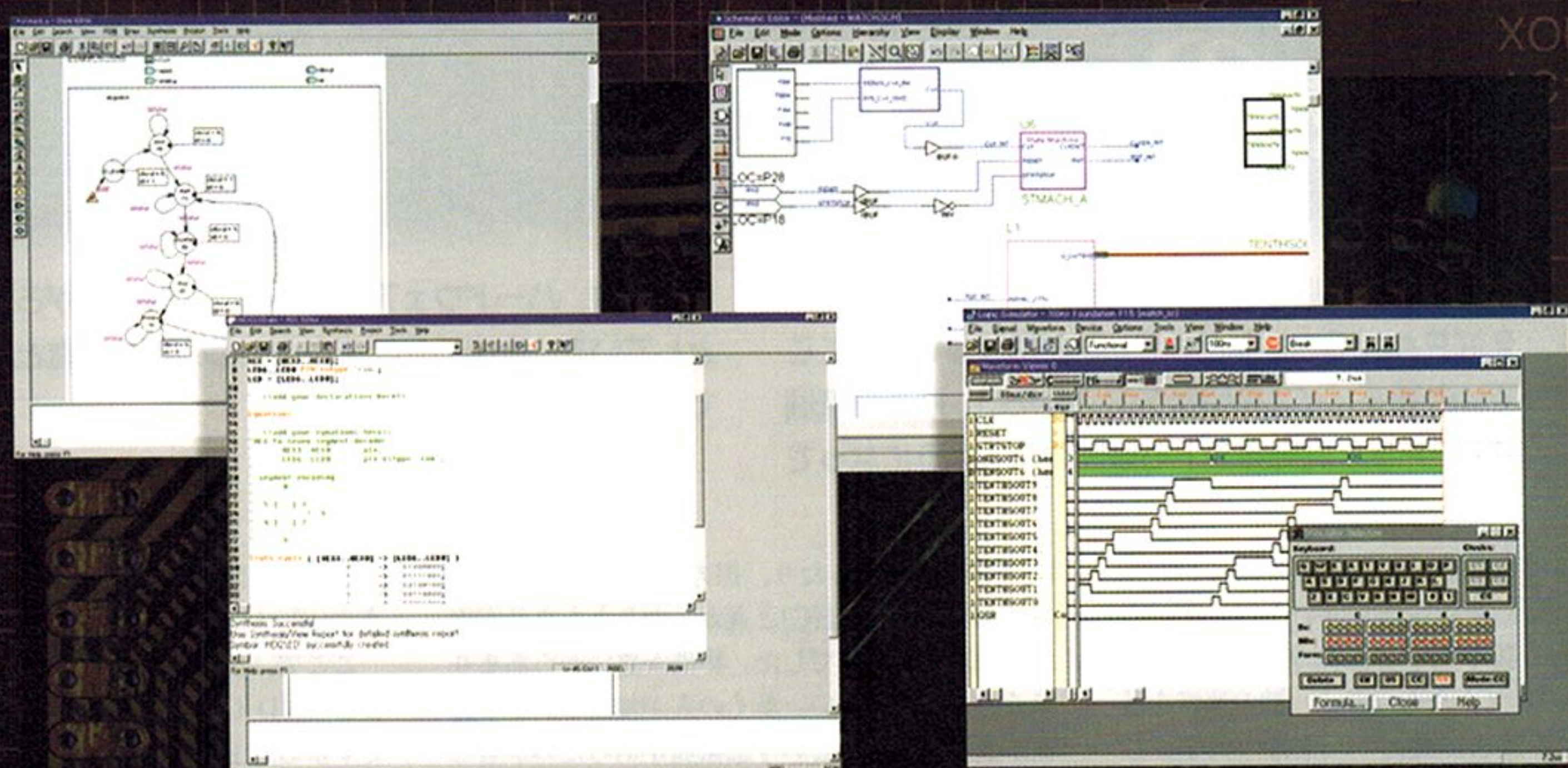
TrackPopupMenu()によって表示されたメニューは，アイテムが選択されると，自動的に第4引数のAfxGetMainWnd()で指定されたウィンドウ，すなわちフレームウィンドウにアイテムIDとともにWM_COMMANDメッセージを発行する。つまり，あとは普通にメニューコマンドのハンドラをインプリメントすればよい。

最後に拡大表示部分だが，ビューのOnDrawをはじめ，ラインの座標取得などにも手を加えることになるが，誌面上では説明しないことにしよう。面倒だがあまり本質的な部分ではないので。各自でリストを見て理解してもらいたい。あと，スケールボタンを単に押した場合，スケールが一段階上がるようにしておいた。

完成である（図10）。ReBarの模様がナンだが，その辺は好きなように変えてCoolに仕上げてもらうといいだろう。ただ，拡大したときのペンがちと遅い。明らかにマウスの後ろから線が遅れてついてきているのがわかる。ちょっと速くマウスを動かすとカクカクする。アドレス計算で手を抜いたせいもあるだろうが，どうやら画面の更新でInvalidateRect()を呼んで，OnDraw()に任せてしまったせいのようだ。まあいちいちメッセージで処理してたらそりゃ遅いわな。描いた部分だけInvalidateしているとはいえ，OnDraw()は全画面描こうとしてるわけだし。

もっと太いペンをつけたりしたら明らかに実用的な速度は得られそうにないので，グラフィックツールを目指す人はちゃんとチューンするように。といったところで，今回は終了。っていうより，EX Pluginの記事に続くって感じかな。

リスト3　メニューの表示

```
void CMainFrame::OnDropDown(NMHDR* pNotifyStruct, LRESULT* pResult)
{
  // ツールバーのドロップダウンで呼ばれるメッセージハンドラ
  NMTOOLBAR* pNMToolBar = (NMTOOLBAR*)pNotifyStruct;
  if( pNMToolBar->iItem==ID_SCALE_DROPDOWN ){
    // メニューの表示座標を、ツールバーのアイテムから取得
    CRect rect;
    m_wndToolBar.GetToolBarCtrl().GetRect( pNMToolBar->iItem, &rect );
    rect.top = rect.bottom;
    ::ClientToScreen( pNMToolBar->hdr.hwndFrom, &rect.TopLeft() );
    // メニューをリソースからロードしてポップアップ
    CMenu menu;
    CMenu* pPopup;
    menu.LoadMenu( IDR_SCALE_POPUP );
    pPopup = menu.GetSubMenu( 0 );
    pPopup->TrackPopupMenu( TPM_LEFTALIGN|TPM_LEFTBUTTON,
    rect.left, rect.top+1, AfxGetMainWnd() );
  }
  *pResult = TBDDRET_DEFAULT;
}
```

特別企画

Device Programing

ハードウェアとソフトウェアは等価だというのは，旧Oh!Xで何度も出てきた概念だった。ソフトウェアでできることはハードウェア化できる，ハードウェアでできることはソフトウェアでも実現できる，ということで，なにかやりたい処理があったときに，それをハードウェアで実現するかソフトウェアで実現するかは実装者次第ということになる。
ハードウェアで実装すると，処理が非常に高速化されるが，逆に開発しにくい，変更しにくいなどというデメリットもある。ソフトウェアの場合，開発や制作後の変更などに柔軟に対応できても，処理速度には限界がある。ソフトウェア処理のデバイスとしてDSPやPICマイコンなどがあるが，これらは論理回路だけで組むと膨大になるレベルの電子工作を劇的にシンプルなものにしてくれる。これらは一般的なCPUと大差ない構造を持つものだが，処理をデバイス内部に押し込むという点では専用コントローラを起こした場合と同等のものと考えることもできる。ハードウェア処理するデバイスとしてはPLD，FPGAなどが挙げられる。これらは専用コントローラと同等の性能を発揮でき，しかも機能の書き換えが可能なのだ。
デバイス内部でハードウェア処理されているか，ソフトウェア処理されているかはユーザーから見れば重要ではない(速度的にクリティカルな場合は除いて)。今回はさまざまなデバイスを紹介したい。
最終的に目指すものは「プログラマブルなコンピュータ」だ。CPUを高速化してすべての処理をソフトウェアで行う，という方向に世の中は移行しつつある。CPUが十分に高速なら，ハードウェア構成をシンプルにできるので，これも悪い選択ではないだろう。一方でハードウェア指向の可能性は見過ごされているようにも思われる。
それはCPU以外の(CPU自体を含んでもいいのだが)コントローラを必要に応じて再定義できるという構成だ。機能の追加はハードウェアの再定義で実現できる。それは，CPUで行う処理をDSPやサブCPUに回す，必要に応じてドライバを組み替えてFPGAを別のコントローラに定義し直す，コントローラの機能自体をリアルタイムに書き換える，さらにはアプリケーションレベルのソフトウェアをハードウェア処理にコンパイルする，といった段階を経て行われることになるだろう。
ソフトウェアとハードウェアの融合を目指して，プログラミングデバイスへの第一歩を始めてみたい。

第1章 石の言葉

高尾 克彦 Takao Katsuhiko

プログラマブルなデバイスでパーソナルなものといえばPAL/GALなどのPLDが挙げられる。ここでは各種デバイスの概要を紹介したい。最近は低価格なFPGA開発キットも販売されるようになってきており，ハードウェア工作の幅も大きく広がろうとしている。今後のPC工作のための基礎知識として押さえておいてほしい。

「無限の知恵に息吹を与えられた書物においては，何一つ偶然に委ねられてはいない。その書に含まれる語の数や文字の配列までもである。そのようにカバリストたちは考え，神の秘密を看破しようとする欲求に燃えて，聖書の文字を数えたり，組み合わせたり，その順序を変えたりする仕事に専念した。(中略)≪ゴーレム≫とは文字の組み合わせによって作られた人間に与えられた名である。この語の文字通りの意味は，無形の，もしくは生命のない土塊である(J・L・ボルヘス，M・ゲレロ『幻獣辞典』柳瀬尚紀訳，晶文社)」

という文章を参考文献1で見つけ，原書を探したのですが見つからなかったので孫引きそのままで始めます。前後の文脈を誤解していたらすみません。

「ある言葉を与えられると特定の動作を始めるもの」といえば，プログラムとコンピュータの関係がまず頭に浮かびますが，そのほかにも，ある言葉を与えると特定の動作を始めるデバイスというのも数多く世の中に存在します。その一例が以下に紹介するPLDというデバイスです。

PLDというものが登場するまでのハードウェア工作の流れを最初にまとめてみましょう。

■TTL(組み合わせ回路)

以前は電子工作というと，図1にあるようなTTLと呼ばれる素子を組み合わせては，あーでもない，こーでもないとハンダづけしていました。

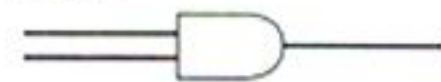
よく皆が使うような回路はバラ売りの素子ではなく，図2-1にあるようなもう少し回路規模の大きいデバイスとして売られていました。これは3本の入力信号に応じて8本の出力信号のうち1本をイネーブルするデバイスで，主にアドレスデコーダに使用されました。たとえば，Z-80では(基本的に)I/O空間が256バイトしかありませんでしたから，8本のアドレス信号のうち，3本をLS138でデコードすることにより，32(＝28－3)バイトごとに区切ることが可能でした。これより細かくメモリ空間を区切ろうとすると，図2-2のようにほかのICを組み合わせなければなりません。

74LS154という4入力→16ピン出力という26ピンパッケージのデコーダもありましたが，極端に入手性が悪かったり，出力確定時間が30[ns](N, LS)，40[ns](HC)と遅かったりとあまり実用的ではありませんでした。特性が悪いからあまり多くの人が使わない。多くの人が使わないから特性が改良されないという悪循環があったのかもしれません。

8ビットのシステムが主流だった時代は，信号線がそんなに多くなかったので，図2-1のような誰かがほしがるような機能は，メーカーがあらかじめ作ったものがあったのですが，16ビットの時代になると扱う信号線が増加し，メーカーもすべての需要にデバイスを供給できなくなります。

また，4入力→16ピン出力の機能のICの場合，4＋16＋2(電源用)＝22ピンのパッケージに入りますが，5入力→32ピン出力のデバイスは5＋32＋2(電源)＝39ピンで，こういうのは縁起もので偶数にしますから40ピン。単にアドレスデコードしたいだけなのに，とても巨大なデバイスになってしまいます。

このように扱う信号線が多くなると，既存のTTLが目的に完全にフィットすることはまず期待できなくなりますから，近いものの組み合わせで，なんとか目的の機能を実現します。

たとえば，16ビットのアドレス空間から，8(＝23)バイトを切り出そうとする場合，16－3＝13本のアドレス信号を見なければなりません。例として，開始アドレスが0xa800(＝0x1010 1000 0000 0???)の場合，A14，A12，A10，A9，A8，A7，A6，A5，A4，A3を反転し，A15～A3のすべてのANDをとる，という作業が必要となります(図3-1)。

ここで，反転させるアドレス線のほうが，そのままAND Gateにつなげるアドレス線より多いですから，ド・モルガンの定義，

!A and !B ＝ !(A or B)

を思い出して，図3-2のように実装したほうがデバイスも少なくて済みますし，ハンダづけも楽そうです。

ここでTTLの規格表を取り出しても，OR Gateにそんな入力数の多いデバイスがないのに気づくだけです。結局，NOTゲートの多さを恨

NOT

入力	出力
0	1
1	0

AND

入力1	入力2	出力
0	0	0
0	1	0
1	0	0
1	1	1

OR

入力1	入力2	出力
0	0	0
0	1	1
1	0	1
1	1	1

NOR

入力1	入力2	出力
0	0	0
0	1	1
1	0	1
1	1	0

図1 TTLの記号

図2-1 74LS138の等価回路図

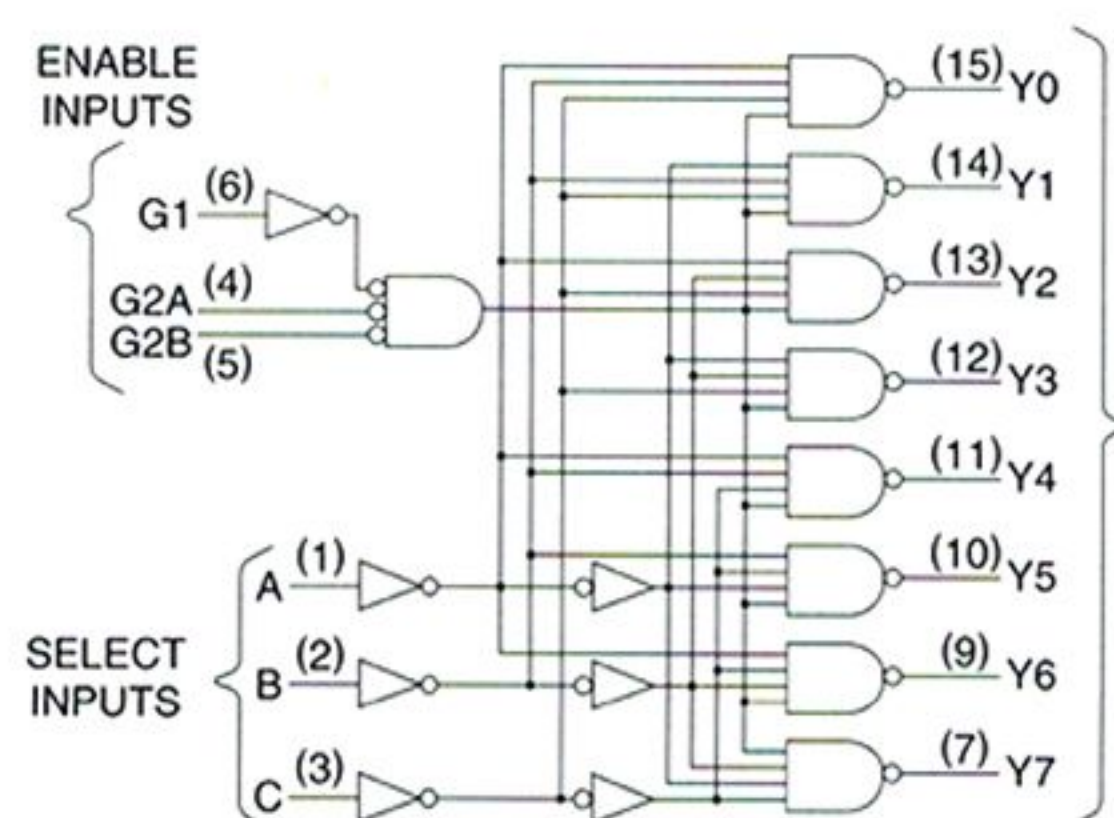



図2-2 0xa800のアドレスデコーダ(その1)

みながら，図3-1をハンダづけを始めます。

このようにTTLを使った回路設計は，まず，必要な機能を書き出し，TTLファミリのなかからいちばん近い機能を持ったデバイスを探し出し，両者を摺り合わせながら行います。規格表に掲載されているデバイスでも，廃品種だったり，入手性が悪かったりCMOS版が出ていなかったりするので気をつけてデバイスを選択しなければなりません。

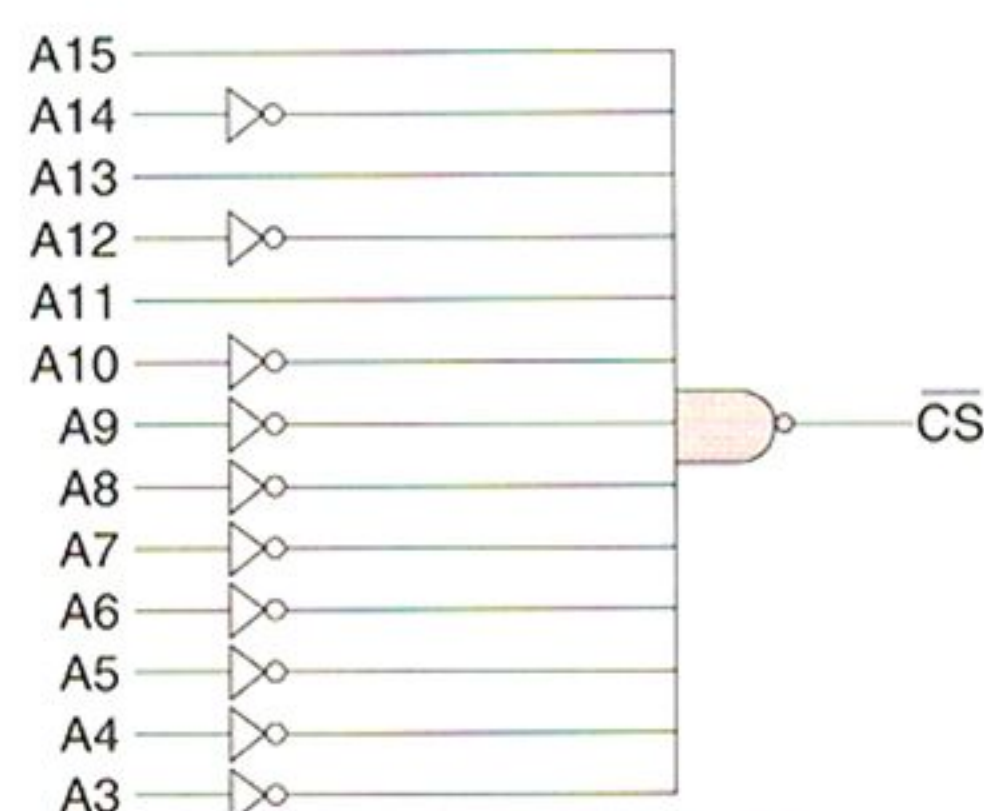
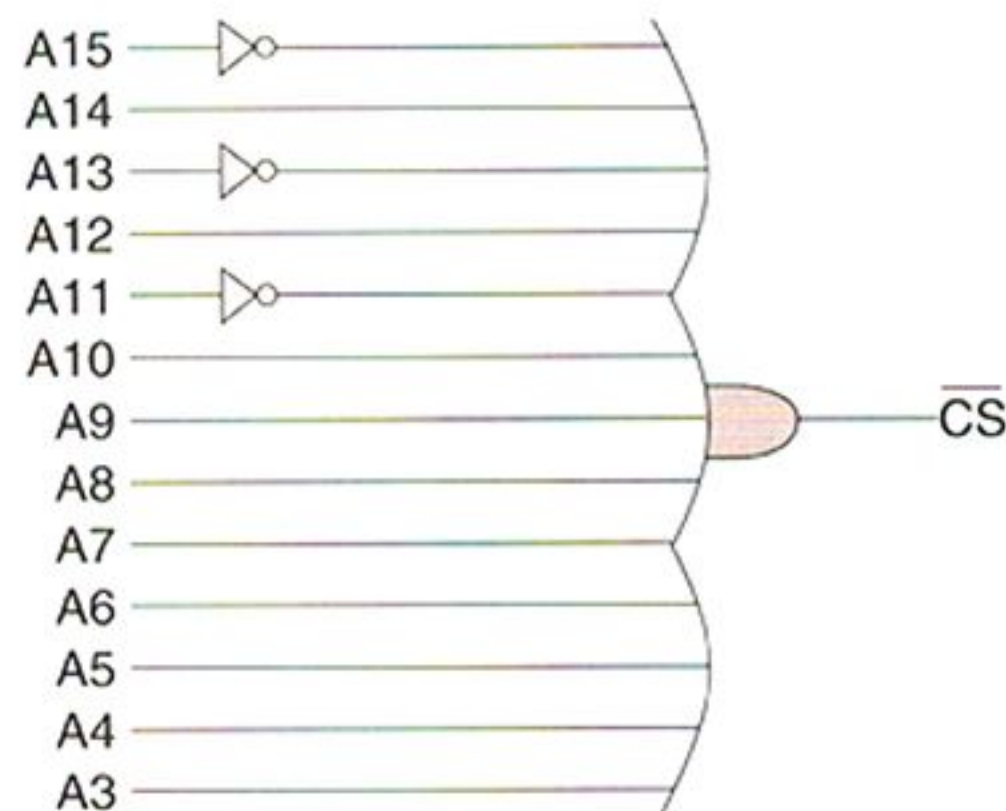
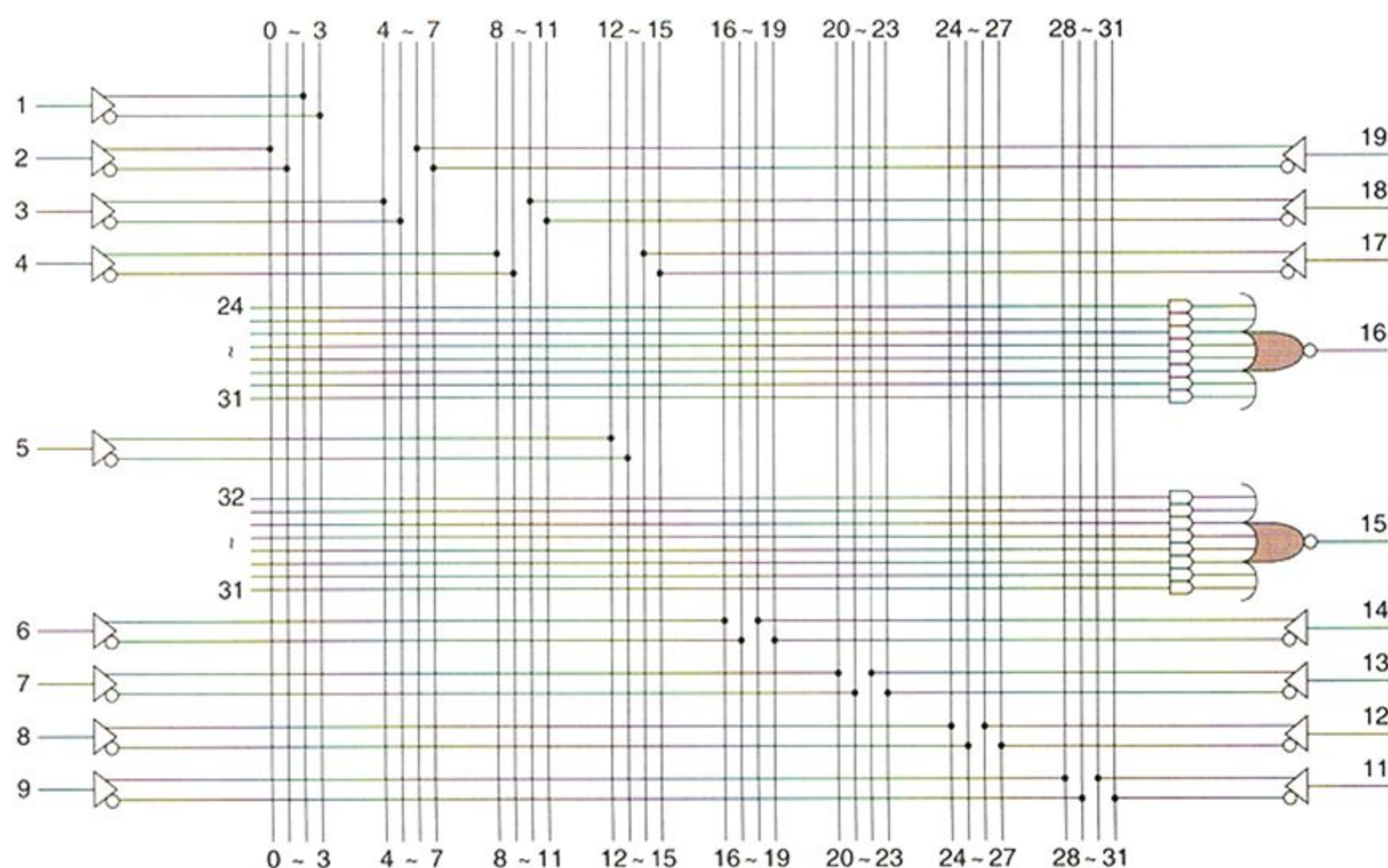
■ROMを用いたロジック回路

TTLによる大規模回路の設計が限界に達し，ここで，ユーザーがカスタムメイドのデバイスを設計できるPLDの登場となるのですが，その間にROMがロジック回路に使われていた時期がありました。

ROMというのは，主にCPUのプログラムを格納する不揮発性メモリですが，あるアドレス信号が入力されたら，対応するメモリに格納されているデータ信号を出力するという動作を行います。これをもう少し柔軟に考えると，格納されるデータはプログラムである必要はなく，また，アドレス信号は，メモリ空間のアドレスを示している必要もないわけです。要は，ある与えられたビットパターン(アドレス線)に対して，出力するビットパターン(データバス線)が一意に決まるというのが，ROMの本質です。

複雑なロジック回路をTTLなどで考えるのではなく，考えられるすべての入力パターンに対して，期待する出力パターンをすべて計算しておきます。この計算結果をROMに焼き込めば，ROMはロジック回路として使用できるのです。

有名な例では，初代IBMPC/ATのアドレスデコーダには74S288という256×8ビットのROMが使用されていました。このROMは単なるアドレスデコーダだけでなく，リアルクロックタイマで使われたり，インテル形式のストローブ信号からモトローラ形式のストローブ信号[1]への変換も行っていました。もともとがROMですので，入力用のアドレス信号数が足りていれば，どのような組み合わせも実現できるのです。

(1)インテル形式とモトローラ形式のストローブ
インテル形式では/IOR，/IOWと，読み出し/書き込み用に別々の信号が割り当てられている。それに対し，モトローラ形式ではストローブ信号は/DSのみで，これが読み出しなのか書き込みなのかはR/W信号で区別する(R/W＝0で書き込み，R/W＝1で読み出し)

■PLD(PAL，GAL)

PALとはProgrammable Array Logic(プログラム可能なロジック回路の配列)の略で，旧MMI(Monolithic Memories Inc.)の登録商標です[2]。また，GALは，Gate Array Logicで，Latticeの登録商標です。一般名詞では，これらのデバイスをPLD (Programmable Logic Device)と呼びますが，PAL(パル)/GAL(ギャル)のほうが呼びやすいので，一般にはPAL/GALという呼び方がされています。

図4-1を見てください。これは，PALのなかでもPAL16L8と呼ばれるデバイスの回路図です。入力付近にバッファが入っており，入力信号の正論理・不論理を，中ほどの編み目のような回路(？)に渡しています。信号はそのあとで巨大なOR回路へと導かれていきます。

一見，図4-1は，なにかを実現する回路に見えなくもないですが，実は，なにも実現しません。これは，いわば「白紙」の状態で，不要な結線を電気的に焼き切ることにより，初めて必要な機能を実現します。

いちばんわかりやすいPALの内部等価回路図を引用しましたが，現在ではPALはほとんど使われません。その代わり，出力段を図4-2のようなマクロセルにして，プログラムによりさまざまな機能を選択できるようになったGALが広く使用されています。また，PALは一度回路を書き込むと変更が不可能ですが，GALは数百回の再書き込みが可能です。

(2)MMIは，K6-IIIで有名なAMD(Advanced Micro Devices)に一時買収されたが，今年からVantisというPLDビジネス専業メーカーとして再び分離・独立した

■CPLD/FPGA

私もこれらの定義がいまだにわからないのですが，どうも，PLDよりもゲート規模の大きいものをこう呼ぶようです。ちなみに，なんの略かというと，CPLD(Complicate PLD)とFPGA(Field Programmable Gate Array)です。どのくらいの規模からいうのかもちょっとわからないのですが，ある特定の出力段を，任意のピンにアサインできる，という機能があるかないかがひとつの目安になっているようです(あまり，自信はないですが……)。

今年の第7回FPGA/PLD Design Conference & Exhibitのテーマは，「ミリオンゲート時代を迎えるFPGA/PLD」だそうですから，いろいろ差し引いても[3]，数十万ゲートのデバイスを設計するのは，もう実用段階に入っているのでしょう。

A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 CS

図3-1 0xa800のアドレスデコーダ(その1)

A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 CS

図3-2 アドレス・デコーダ(その2)

0～3 4～7 8～11 12～15 16～19 20～23 24～27 28～31
1 2 3 4 5 6 7 8 9 19 18 17 16 15 14 13 12 11 24 31 32 31

図4-1 0xa800のアドレスデコーダ(その1)

これらの機能を使用するために，それなりにお金を掛けられたとしても，

ゲート数が多い
　→機能が多い
　→ピン数が多い
　→パッケージが特殊＆ピン間隔が狭い
　→ハンダづけ不可能

というアマチュア特有の問題点もあります。

個人的には，アマチュアレベルでは，1～2万ゲートくらいが限界かなと思っています。

(3)PLDは編み目を焼き切って，必要な回路を炙り出すという構成のため，炙り出したあとには未使用になってしまうゲートがかなり存在します。通常は，40～60%くらいが使えません。たとえば，1000ゲート内蔵のPLDに対しては，400～600ゲート規模の回路しかインプリメンテーションすることができません。あとは，マーケティング計数とかあるかもしれませんね。

■HDL

PALの黎明期には，図4-1の下方に小さく書いてある数値を基に，結線を焼き切る機械(PALライタとかプログラムライタという)に動作情報を与えていたようです。この動作情報は，PLD内の結線を焼き切るためのものですから，ヒューズパターンと呼ばれています。また，ヒューズパターンのフォーマットはわりと古くからJEDEC

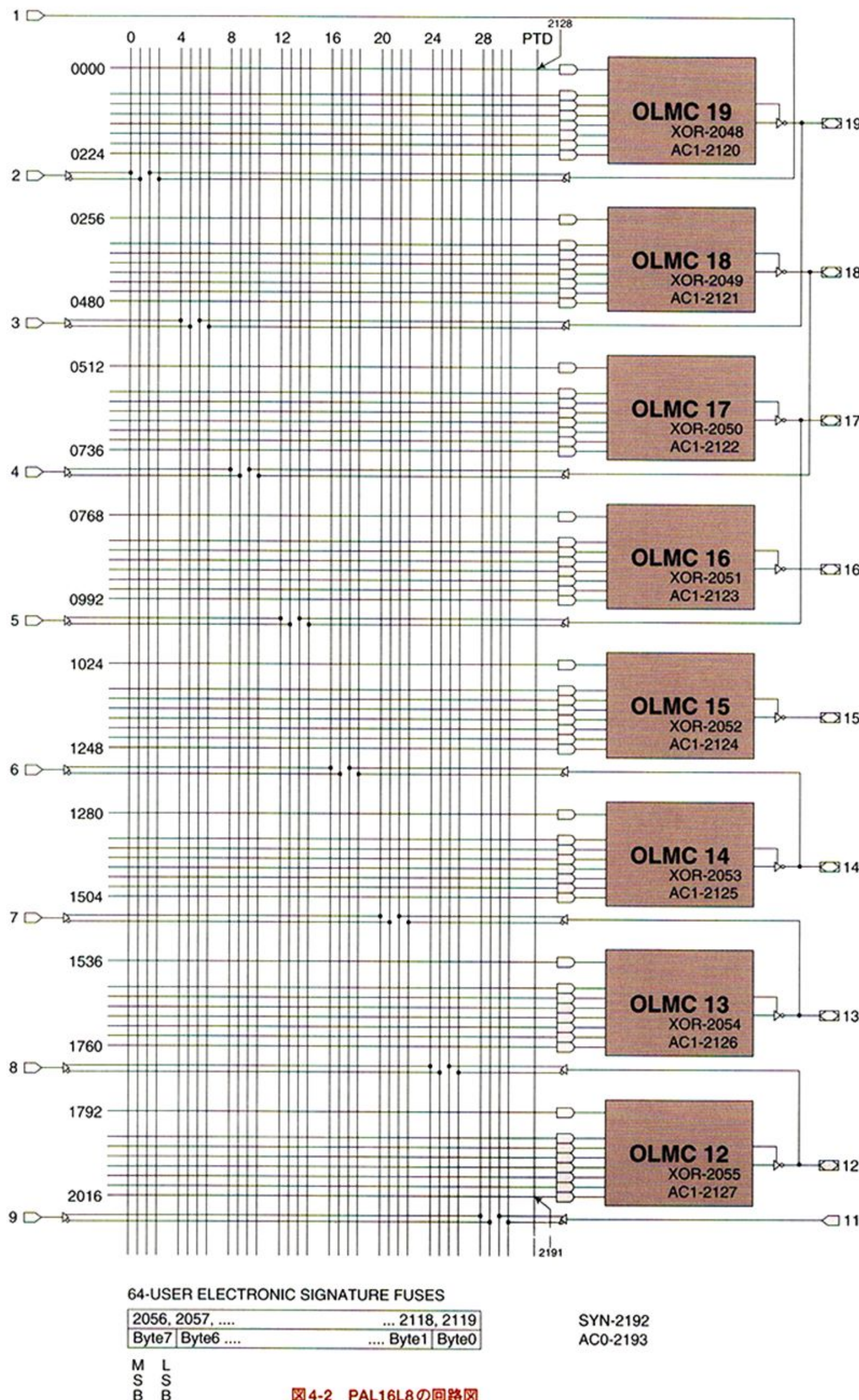
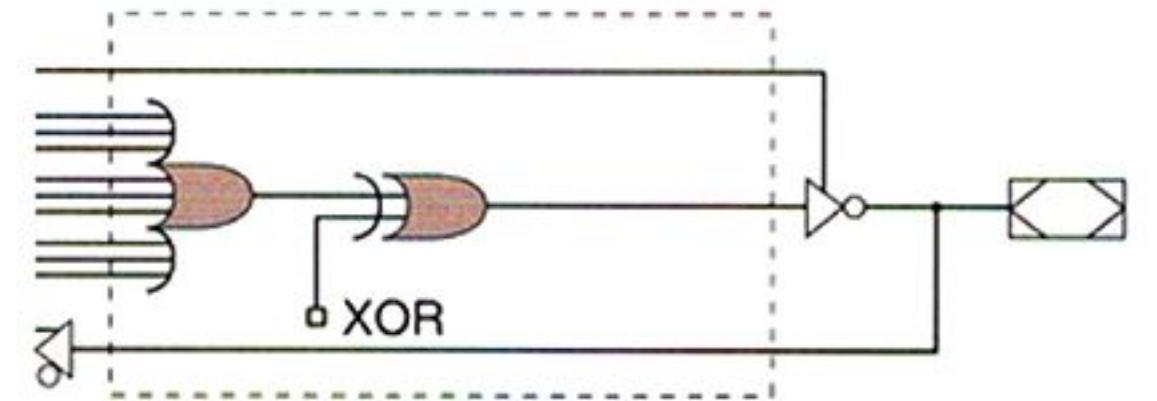



図4-2　PAL16L8の回路図

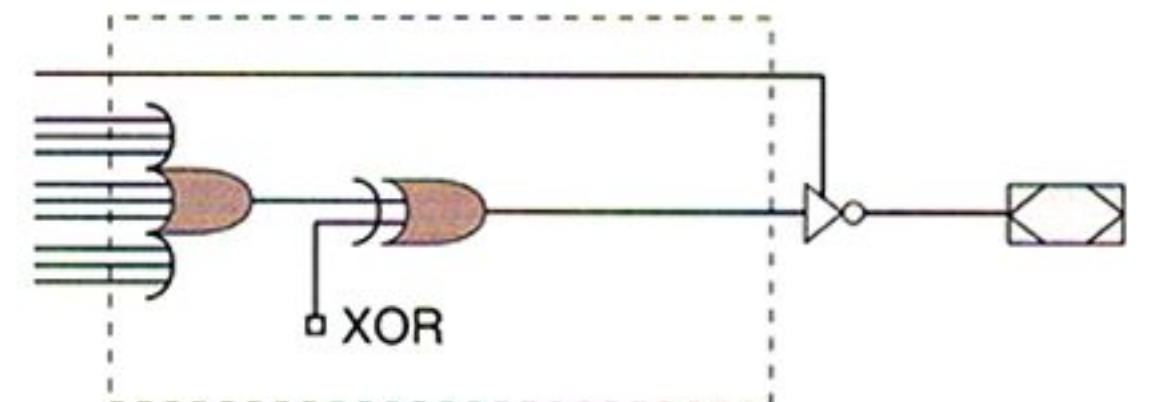
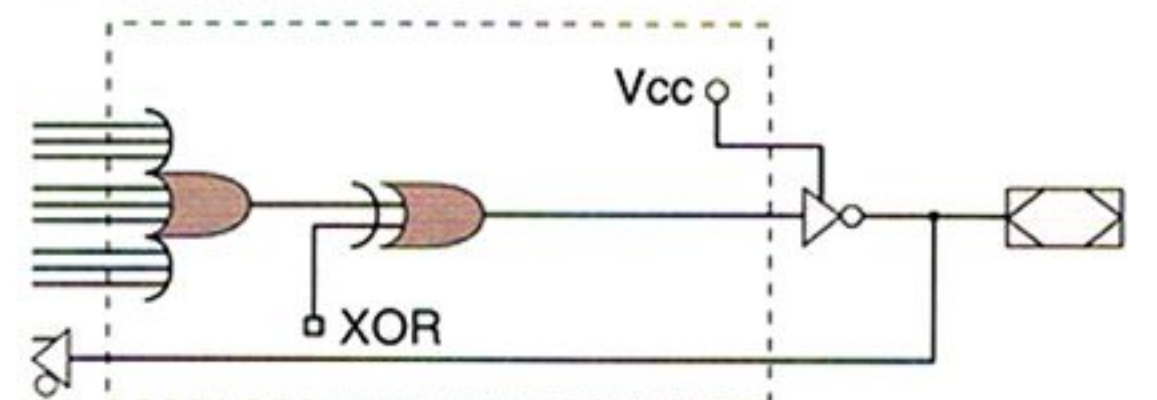
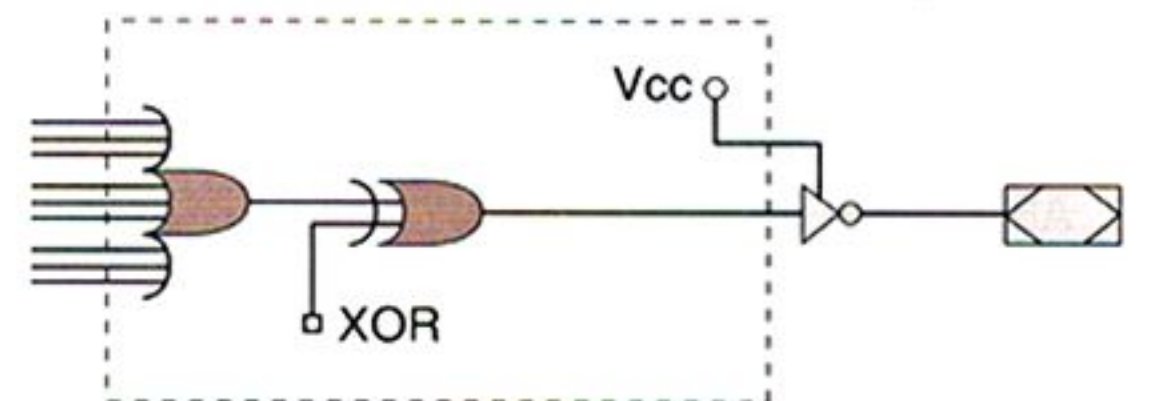
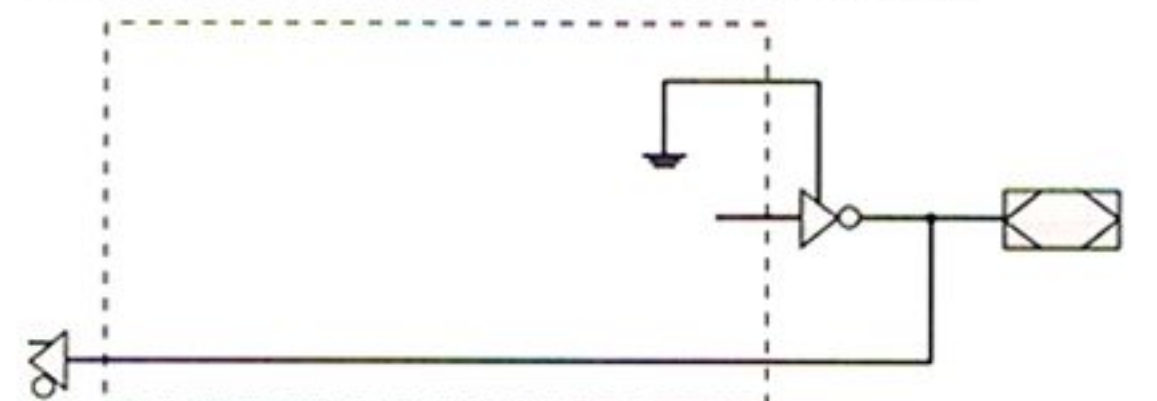
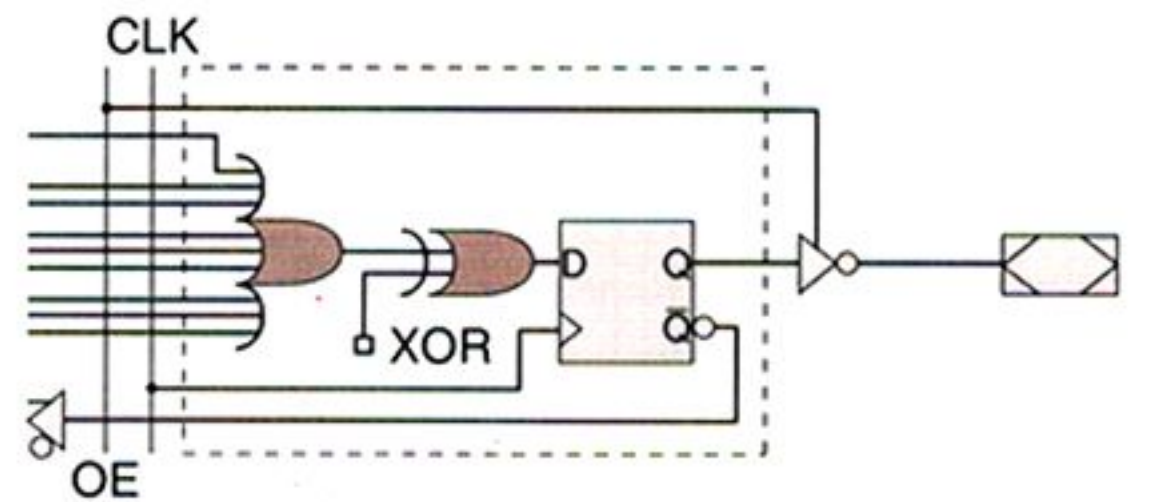
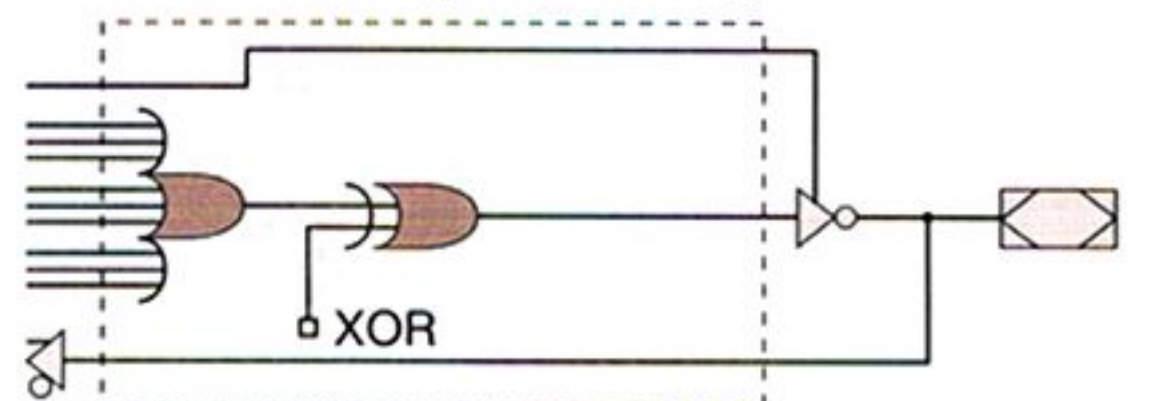



図4-3　GAL16V8の回路図・出力ロジックマクロセル

(Joint Electronics Device Engineering Council)という組織で制定されていて，その名前をとってJEDECファイルと呼ぶ場合もあります。

これは，彼らの用語でも「ハンドコンパイル」といっていたくらい，一般人には理解しがたいものでした(基本的に16進数の羅列です)。

そこで，高級言語が登場します。

「！」：反転
「＋」：論理和
「＊」：論理積
「＾」：排他的論理和

を用いて動作を記述すれば，自分だけのオリジナルデバイスを作ることができます。プログラミング言語にも，C言語やBASICなどと，各種あるようにHDLにも何種類かあり，ものによっては，

「／」：反転
「｜」：論理和
「&」：論理積
「＾」：排他的論理和

と記述する場合もあります。

それはともかく，このようにPLDにどのような動作を期待するかを記述するための言語をHDL(Hardware Description Language)といいます。

このHDLを用いると，先ほどの16ビットのアドレス空間から，0xa800(＝0x1010)を先頭アドレスとして8(＝2^3)バイトを切り出す例は，

!CS＝A15＊!A14＊A13＊/A12＊A11＊!A10＊!A9＊A8＊!A7＊!A6＊!A5＊!A4＊!A3

と記述できます。

PAL16L8を用いる場合には，このデバイスの制限として，論理和は8つまでという制限がありますから，

!CS0＝A15＊!A14＊A13＊!A12＊A11＊!A10＊!A9
!CS1＝!A8＊!A7＊!A6＊!A5＊!A4＊!A3

のように，上位アドレス(!CS0)・下位アドレス(!CS1)の2つに分けて，これらの出力の論理積をとり，実際のチップセレクト信号とします。

!CS＝!CS0＊!CS1

ここで，鋭い方は，いくつかの疑問を抱くかもしれません。

「n A15やA14とは，どのピンか？」
「図4-1には，OR Gateしかないが，なぜ，論理積が使えるのか？」

最初の質問の答えは，設計者が命名します。マシン語のラベルと同じで，HDLでも，ラベル機能が使えます。この機能を使えば，デバイスの2番ピンを「A15」という名前に，3番ピンを「A14」という名前にと，その信号の名前を扱えるようになります。これは，論理式を記述しやすくするだけでなく，たとえば，基板を設計していて，2番ピンと3番ピンの信号を交換したくなったときでも，すべての論理式を見直すのではなく，このラベルを付け直すだけで，対応できるという利点があります。

2番目の疑問に対する答えは，コンパイラが面倒を見てくれるからです。ド・モルガンの法則により，

A＝B＊C
　＝!(!B＋!C)

ですから，

!A＝!B＋!C

です。つまり，入力バッファと出力バッファの極性を反転できる機能があれば，ORゲートだけでも，ANDゲートが実現できるのです。また，XORゲートもデバイス自身にはありませんが，論理値表の，

入力A	入力B	出力
0	0	0
0	1	1
1	0	1
1	1	0

のとおり，

A＾B＝(A＊!B)＋(!A＊B)

のように展開されます(一般のデバイスでは，排他的論理和は，単純な論理和・論理積の倍のリソースを使うので，なるべく使わないようにします)。

さらに，HDLコンパイラは，記述→機能の翻訳だけでなく，論理圧縮と呼ばれる最適化作業も行います。

アドレスデコーダくらいの複雑さでしたら，記述に冗長性はほとんどありませんが，たとえば，図5のような7セグメントLEDを操作する場合，「A」と書かれた部分は，0，2，3，5，6，7，8，9のいずれか(OR)のときに光るので，

```
LED_A＝(!D3＊!D2＊!D1＊!D0) /＊0＊/
  ＋(!D3＊!D2＊D1＊!D0)     /＊2＊/
  ＋(!D3＊!D2＊D1＊D0)      /＊3＊/
  ＋(!D3＊D2＊!D1＊D0)      /＊5＊/
  ＋(!D3＊D2＊D1＊!D0)      /＊6＊/
  ＋(!D3＊D2＊D1＊D0)       /＊7＊/
  ＋( D3＊!D2＊!D1＊!D0)    /＊8＊/
  ＋( D3＊!D2＊!D1＊D0)     /＊9＊/
```

と記述できます。ここで，上式を見ると，

```
  ＋(!D3＊!D2＊D1＊!D0) /＊2＊/
  ＋(!D3＊D2＊D1＊!D0)  /＊6＊/
```

D2の値に関わらず，

D3＝0，D1＝1.D0＝0

のとき(入力が2か6)，光るので，

```
LED_A
＝(!D3＊!D2＊!D1＊!D0)  /＊0＊/
＋(!D3＊!D2＊D1＊!D0)   /＊2＊/
＋(!D3＊!D2＊D1＊D0)    /＊3＊/
＋(!D3＊D2＊!D1＊D0)    /＊5＊/
＋(!D3＊D1＊!D0)        /＊2/6＊/
＋(!D3＊D2＊D1＊D0)     /＊7＊/
＋( D3＊!D2＊!D1＊!D0)  /＊8＊/
＋( D3＊!D2＊!D1＊D0)   /＊9＊/
```

と記述できます。また，0～9以外の入力がないと保証されている場合には，D3の値に関わらず，

D3 ＝ 0，D1 ＝ 0，D0 ＝ 0

のとき(入力が0か8)，光るので，

```
LED_A
＝ (!D3＊!D2＊D1＊!D0)  /＊2＊/
＋(!D3＊!D2＊D1＊D0)    /＊3＊/
＋(!D3＊D2＊!D1＊D0)    /＊5＊/
＋(!D3＊D1＊!D0)        /＊2，6＊/
＋(!D3＊D2＊D1＊D0)     /＊7＊/
＋(!D2＊!D1＊!D0)       /＊0，8＊/
＋( D3＊!D2＊!D1＊D0)   /＊9＊/
```

と記述しても同じことです。

このような論理式の最適化を行っていくと，うまくすれば，もっと安いデバイス(ゲート数が少ない)で実現ができたり，動作スピードを上げることができたりと，いろいろな特典があります[4]。

しかし，この作業を人間がやるのはとても面倒ですし，不要なバグのもとです。大本の仕様からかけ離れてプログラムが読みづらくなってしまいます。

そこで，たいていのHDLコンパイラには，このような論理式の最適化作業を行う機能がついています。どの程度まで最適化を行うかはコンパイラにもよりますが，数秒で終わる範囲のものから半日近く考え込むようなものまで，幅広くあります。

以上，大まかなHDLの説明をしてきましたが，PLDには，ほかにもラッチ回路やトライステート機能[5]などがあるものがあり，そのような細かい

D3
D2
D1
D0
a b c d e f g
0 1 2 3 4 5 6 7 8 9

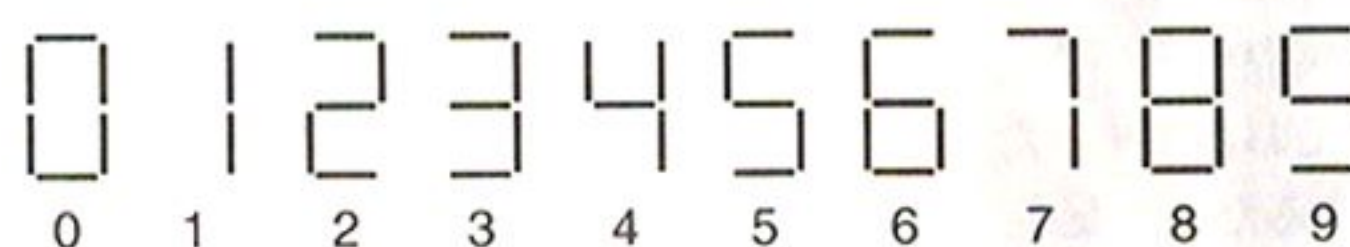
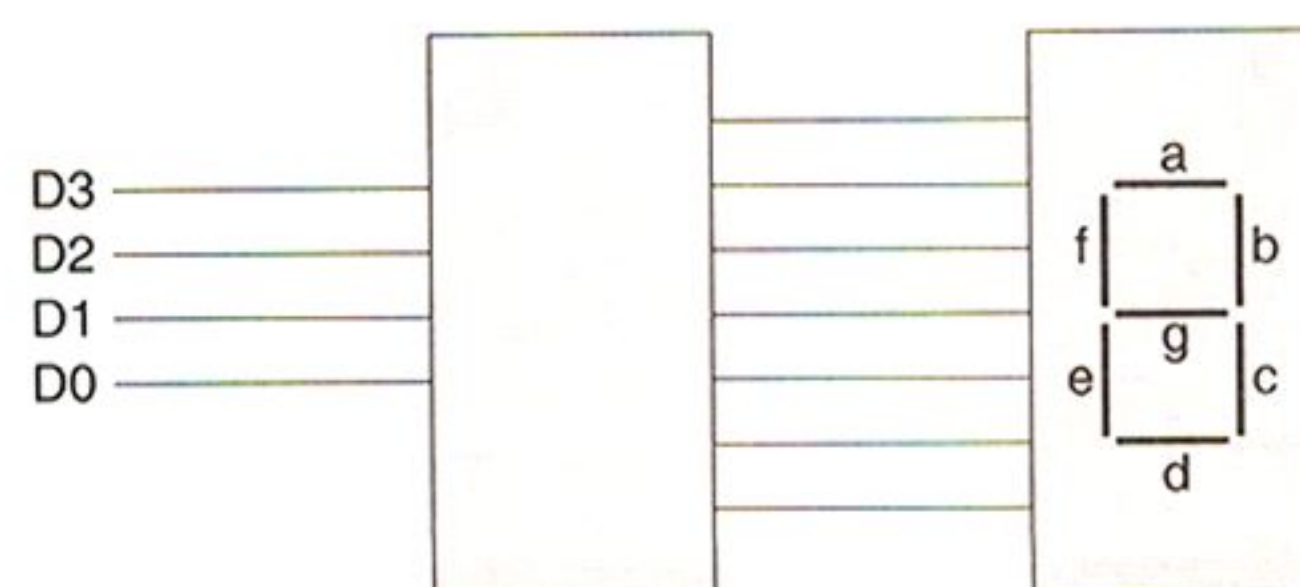



図5　7セグメントLEDの制御

指定は各種HDLで微妙に異なってきます。また，疑似命令として，先ほどのアドレスデコーダを自動的に生成する機能やピン配置を最適化する機能を持ったものがあります。

現在，このようなHDLとして，

MACHXDL(PALASM)
CUPL
ABLE
Verilog-HDL
VHDL

などがあります(した)が，詳しい話はまた別の機会にゆずります。

(4)仕様を逆に見て，「A」と書かれた部分は「いつ点灯するか」ではなく，「1と4のとき点灯しない」と解釈すれば，

!LED_A＝(!D0＊!D1＊!D2＊D3)
　＋(!D0＊D1＊!D2＊!D3)

と2項で書けるが，通常，PLDコンパイラはここまで最適化を行わないのでプログラマの腕の見せどころとなっている。

(5)別名、ハイインピーダンス状態。0[V]も5[V]も出力しない状態のこと。データバスなどのように複数のデバイスが共有するような信号線で，自分が発言すべきでない場合は，この状態にしておき，発言者のじゃまにならないようにする。たとえば，通常，コンピュータシステムではCPUがデータバスを駆動するが，DMA動作中はDMAコントローラがデータバスを駆動する。その際，CPUのデータバスピンはDMAの動作を邪魔しないようにトライステート状態になる

■シミュレータ

TTLで組んでいた回路をPLDで組むと，デバイス数が減る，高級言語で記述できる，機能の変更が容易，などのメリットが得られるわけですが，さらにもうひとつ大きな利点があります。

デバイスの機能をHDLで記述し，コンパイラにかける，ということは，コンピュータはHDLを理解できるということです。ですから，HDLからヒューズパターンを生成するだけでなく，HDLをシミュレートすることもできます。特に回路規模が大きくなってくると一発で動作させることはかなり難しいのでトライ＆エラーを繰り返すのですが，そのたびにいちいちデバイスを基板から取り出してヒューズパターンを焼き込み，再び基板への装着を行っていたのではたいへん手間です。それに，いちいち各ピンへの入力を変えてみて出力を見るというのも手間がかかります。

そこで，このような検証作業は実際のデバイスを焼き込む前にPCで十分に行っておきます。たいていのHDLには論理式を記述するだけでなく，コンパイル終了後にどのような検証を行うかというテストベクタを記述できます。この段階で「＊」と「＋」の打ち違いや「！」を忘れたなどの簡単なミスは，ここでエラーとなりますので，簡単に発見することができます。

これらのテストベクタは，デバイス書き込み後に行うベリフィケーションに用いることもできますので，ヒューズパターンが確実にデバイスに書き込まれたかを確認する際にも役立ちます。

最近では，ソースファイルに記述するバッチ式のものだけでなく，GUIを用いて対話的にシミュレーションを行うものもあります。

■ステートマシン

いままであまり詳しく説明してきませんでしたが，PLDでは出力段にDフリップフロップなどを指定することができます。このバッファは，出力変化をクロックが与えられるまで待ちます。というわけで，PLDでクロックに同期するカウンタ回路を作るのはわけもないことです。

カウンタ値をデコードするのは，いままで見てきたようにまったく問題ありませんから，たとえば，

0：左・右のモーターを回す
1：左のモーターのみを回す
2：目を光らせる
3：右のモーターのみを回す
以上，繰り返し。

というようなシーケンサも簡単に組むことができます。

さらに，「カウンタの値を適宜更新してはいけない」というルールはどこにもありませんから，シーケンスの順序を意図的に変更することもできます(1→2→3→4ではなく，1→3→4→2など)。また，アドレス情報以外にも，センサの出力もデコーダに入力できますので，

0：左・右のモーターを回す
1：左に障害物があれば3へ
2：左のモーターのみを回す
3：5へ
4：右のモーターのみを回す
5：目を光らせる
6：0へ

のような条件判断を加えることも可能です。この各状態のことを「ステート」と呼ぶので，このような回路は，ステートマシンと呼ばれることもあります。

このような機能を使えば，簡単な処理はCPUを使わずに，PLDだけで処理することも可能です。たとえば，PS/2マウスには，小型のCPUが入っていますが，この程度の処理ならPLDで置き換えることも可能です。

■プログラムライタ

さて，HDLで新しい回路を設計し，シミュレートの結果，十分に満足のいく設計ができたとしましょう。これらの結果は，すべてPCの内部で起きていることですから，なんらかの方法で実際のデバイスに伝える必要があります。

この点に関して，PLDには以下の2種類があります。

●PLD自身に書き込むもの

従来のPAL/GALなどがこのタイプで，専用のプログラムライタを用いて焼き込みます。PALの説明の項で，網目状の結線のうち，不要なものを焼き切るといいましたが，この作業自体は，EEPROMの書き込みとわりと似ています。ですから，プログラムライタの外見も，そっくりですし，なかには，EEPROMとPLDの両方を書き込める，というプログラムライタもあります。こういうものは何百台と売れるものではないわりに，代理店が二重三重に入っていたりしてたいへん高価です。

また，プログラムを書き直すたびにデバイスを基板から取り出さなければならないので，ソケットを使用しなければなりません。ソケットは基板上の場所をとり，周波数特性も問題になる場合があるので，システムの小型化や高速化を図る場合には不利です。また，デバイスのピン数が100を超えると，対応するソケットすらないという問題もあります。

●PLD自身には書き込まない

これは，FPGAによくある話ですが，デバイスが電源投入時に外部メモリ(たいていは，シリアル接続されたEEPROM)から，ヒューズデータを読み込みます。ブートするという人もいます(PCが電源投入時にHDDからOSイメージを読みに行くのもブートですね)。ヒューズデータは，たいていRAM内にコピーされて保存されますの

FPGA　CPU　フラッシュメモリ
PCのプリンタインタフェイスへ
TDO　TMS　TCK　TDI

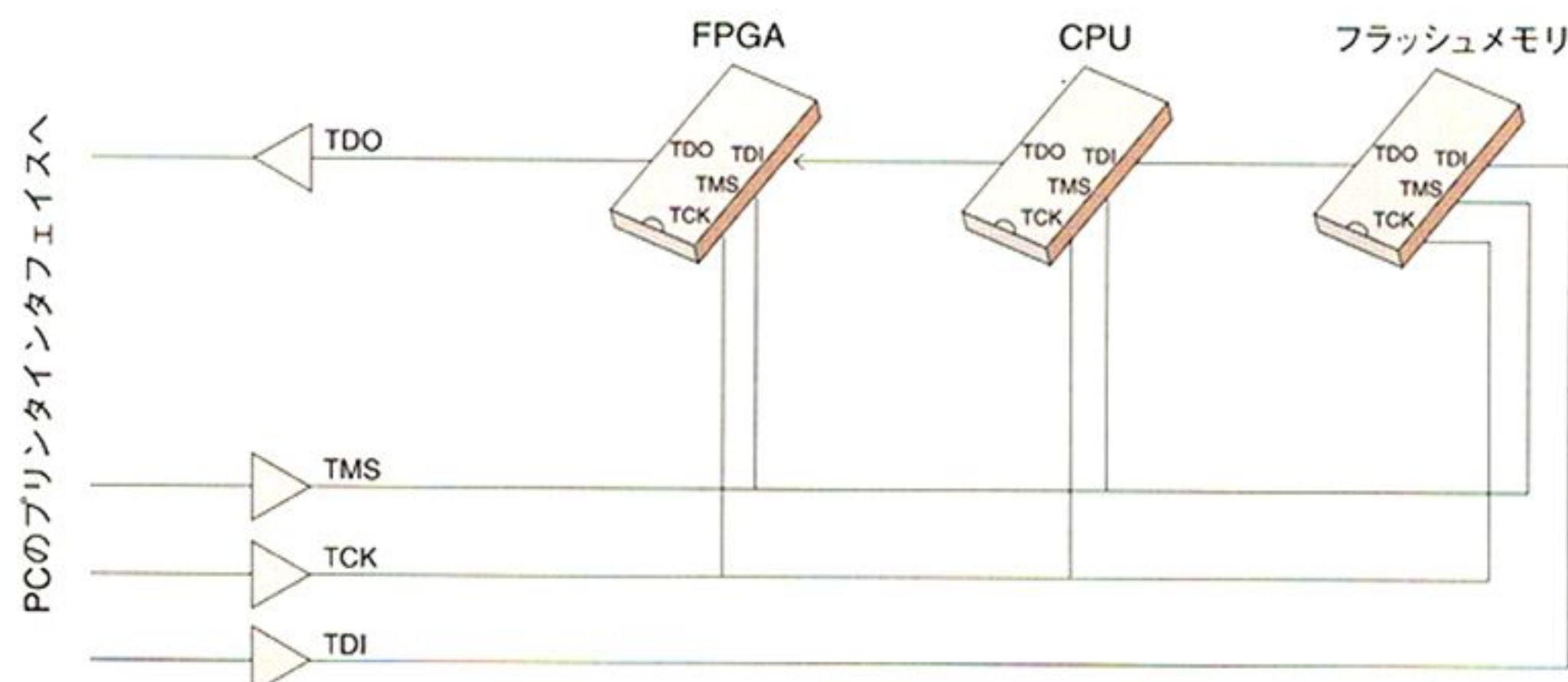



図6　JTAG信号のデイジーチェーン

で，EEPROM型のメモリを内蔵しているPLDに比べて，より高速動作が可能，低消費電力などのメリットがあります。

●ISP(In System Programing)

ISPとは，その名のとおり，システム内のデバイスにプログラミングを行う機能のことです。この機能を用いると，デバイスを基板から取り出さずに，プログラミングを行えます。

ISP対応のデバイスには，たいていISP専用のシリアル信号が装備されており，そこからヒューズパターンを書き込むことができるようになっています。シリアル信号ですから，クロック，シンク(同期)，データなど数本で，タイミングを作成するのもそんなに難しくありません。というわけで，高価な書き込み機も必要なくなりたいへん便利です。たいていは，PCのプリンタポートから信号線を何本か引っ張ってくればそれで十分です。

最近はJTAG(Joint Test Action Group IEEE 1149.1で規定)という信号をISPに使用するデバイスも増えてきました。このJTAG信号というのは，本来プリント基板実装時の，デバイス→基板→デバイス結線テスト用に規格化されたのですが，結線チェック以外にも各デバイスメーカーが独自の機能を盛り込めるようになっていて，PLDではその機能をISPに使用しています。

また，この規格はデバイス間のデイジーチェーンをサポートしていますので，PLDごとにISP用ソケットを設ける必要もなく，図6のように基板に1カ所ソケットを設ければすべてのデバイスをカバーすることができます。

最近はFPGAだけでなく，フラッシュメモリもJTAG経由による書き込みをサポートしていますし，組み込みマイコンもJTAG経由によるリモートデバッグ機能をサポートしているものがありますので，5本のJTAG信号さえ立てておけばすべてのリソースにアクセスできるというようなシステムも，今日では十分に実現可能です。

■終わりに

かなり大まかにですが，FPGA/PLDの概要を説明しました。このようにアマチュアでも数千ゲート規模のデバイスを作成できる時代になりました。

具体的な話をあまりしませんでしたが，これは，筆者の力量のなさに加え，具体的なプラットフォームを想定しないと，かえってわかりづらくなってしまうと考えたからです。

本誌2号を見ると，満開製作所「零式」にはPCIブリッジや「Qバス」インタフェイスなどに積極的にFPGAを使用するみたいですので，そちらで具体的な話をいろいろうかがうことができるでしょう。プログラミングでも同じですが，こういうのは具体例を見ながら少しずつ自分なりの改造を加えていくのが習得のいちばんの近道だと思っています。「零式」の記事が入門には難しすぎた場合にこの記事を思い出していただければと思います。

Boxed Article

PCI

CPUがまだi386やi486で，25MHzや50MHzで動いていた頃はPCの拡張基板を自分で作るといっても，アドレスデコーダを作ってデータバスを直結すれば，たいていのインテル系周辺デバイスは動作しました。これはISAという基板コネクタの規格が，CPUの信号をほぼそのまま出力していたためです。

バス規格がISAから，PCIに移った現在，単にI/O信号を外部に出力する拡張カードを作ることさえ難しくなりました。ISAに対して，ピン間が細くなった，動作速度が8MHzから33MHzと4倍になったなど，性能を上げた代わりにアマチュアの回路製作を難しくしている点はいくつかありますが，いちばんの難関はPlug&Playの実装でしょう。Plug&Playとは，ある新しい拡張カードをPCに入れてWindows98などを立ち上げると「不明なデバイスが検出されました」といって，デバイスドライバのありかを聞いてくるあの機能です。逆に拡張カードを抜いた場合にはデバイスドライバのロードをやめます。

PCI拡張カードには，通常のデータをやりとりするメモリ空間とは別に，PCIコンフィグレーション空間と呼ばれる，特殊な空間があります。一般のメモリ空間がメモリアドレスをデコードして選択されるのに対し，このPCIコンフィグレーション空間では，スロット番号で区別されます。Windows98は，起動時にPCIスロットを順に調べ，新しい拡張カードを検出すると「不明なデバイスが……」と画面に表示します。

このPCIコンフィグレーションからは，使用されているデバイス，ボードメーカーの種類などが，ID値として読み込めます。たとえば，Voodoo Bansheeは，デバイスがxxxx-xxxxxで，基板のIDがxxxx-xxxxです。このデバイス番号の上位16ビットは，半導体メーカーのIDで，基本的に通し番号になっています(なぜかインテルは，0x8086を予約していますが)。

PCIコンフィグレーション空間は，読み出しのほかに書き込みもできます。この機能を利用して，このカードをI/Oアドレスのどこに配置するか，どの割り込みを使用するか，などを指定します。

これらの機能がなかった頃は，ユーザーがすべてのカードのI/Oアドレスや割り込み信号をぶつからないように計算してジャンパ設定を行っていました。これは，かなりの経験者が行っても問題の多発する箇所でしたから，ここを自動化するのは，PCを大衆製品として売るためには必要不可欠なことだったのでしょう。

一般ユーザーにはよいことずくめのPCIバスですが，この機能を設計するのはとてもたいへんです。どんなカードを設計するにも，以上の機能をまず実装しなければなりません(さらに厳密にいうと，先ほどの通し番号をPCI-SIGという公的機関に申請して自分だけのIDをもらう必要がある)。参考文献2)によると，

- これらの最低限の機能＝約5000ゲート
- マスタ機能やバースト転送をサポートする場合＝約1万～3万ゲート

必要とのことですから，もう，TTLで組むのは実用的ではありません(ちなみに，Z80が約4000ゲートだそうです。また，真偽のほどはわかりませんが，モトローラのMC68000は，68000トランジスタ(≒17000ゲート？)という出所のわからない噂もよく聞きした。Plug&Playを実現するために，昔のCPUよりも大規模な回路が使用されていると，感慨深いものがあります)。また，Tritonチップセットのなかには，Plug&Play BIOSが不必要にバースト転送を行うものもあったらしく，最低限の機能だけのPCIカードでは受け付けてくれないPCも世の中にはあるみたいです。

ですから，最近ではアマチュアの拡張カードは，たとえ機能がI/O出力だけでもFPGAを使うことが多くなってきました。また，FPGAメーカーからも，ライブラリとしてPCIインタフェイスを実現するVHDLを(たいていは有償で)公開しているようです。

(注)Plug&PlayはPCIの一部であって，PCI＝Plug&Playではありません。また，Plug&PlayはPCI以外のインタフェイスにも使われていて，ISAでも(無理やり)実装することもできますし，PCMCIAカードのPlug&Playでは，カードの名前が検出できるようになっています。最近では，USB用のPlug&Playもあります。

参考文献

(1)西垣 通「デジタルナルシス」岩波書店 1991
(2)トランジスタ技術スペシャル「PCIバスの基礎と応用」CQ出版社，1999

Column

Xilinx Foundation BASIC

本文中では，各ステップの実例をあまり挙げず，少し抽象的な話が多くなってしまいました。FPGA/CPLDの開発ツールは，それぞれアクが強く，あまり操作が統一されていません。本文である特定のツールを用いると，ツールの説明をしているのか，FPGA/CPLDの説明をしているのかわからないような記述になってしまうのを避けるため，あえて具体的なツールの説明を省いてみました。

とはいっても，抽象的な話ばかりでは実感がわきづらいですから，ここではXilinxのFoundationというツールを使用してFPGA開発の大まかな流れを紹介します。

■回路の入力

まず，回路の設計ですが，この開発ツールは以下の3種類の方法を用いることができます。

●回路図入力(図3)

TTLを使った設計を行ったことがあれば，回路図入力でFPGAを設計するのはそんなに難しくないでしょう。基本的に専用のドローツールで回路図を書き込み，できあがったらコンパイルするというスタイルです。このエディタは階層構造をサポートしていますので，下の階層でAND/OR/D-Latchといった基本的なモジュールを作っておけば，上位の階層ではそのモジュールをあたかもひとつの部品のように扱うこともできます。

たとえば，本文中で紹介したバイナリコードを7セグメントに表示するモジュール(FoundationではMacroと呼ぶ)を下位のレイヤーで作成し，上位のレイヤーではあたかもそのような部品があるかのように配置することができます。

●HDL入力(図4)

HDLは，単なるテキストファイルをコンパイラに渡すだけですから自分の気に入りのエディタを使ってもよいのですが，この開発ツールに付属のエディタには汎用のテキストエディタにはないHDL記述に便利な機能がついています。

予約語を色づけして表示するのは当然として，テンプレートを自動生成する機能や，ライブラリからもっとも近い機能を引用してくる機能などがありとても便利になっています。入門キットで使用できるHDLはABELのみとなっていますが，VHDLやVerilog-HDLが使用できるバージョンもあります。

●状態遷移図入力(図5)

フローチャート作成プログラム(Mac Flowって覚えてる？)みたいな感じで，状態遷移図を作成するとABELかVHDLのソースを自動的に生成します。

前者2つの方法にエディタに比べて，使い勝手がいまひとつですが，処理によってはそれでも状態遷移入力を用いたほうが便利という場合もあるかもしれません(機械制御が専門でデジタル回路の設計やプログラミングの経験がほとんどない人は，直感的な状態遷移図がいちばんわかりやすいでしょう)。

■シミュレーション

作成した回路は実際の基板に実装する前にシミュレーションを行い，間違いがないかよく検証します。シミュレーションは，それぞれの信号線の状態を表示させてみるのですが(図6)，このときに入力信号をクロックや式で指定するだけでなく，キーボードのキーで対話的に制御することも可能です。たとえば，「R」キーを回路のRESET信号に割り当て，「R」キーを押すたびに，回路にリセットがかかるというようなことも実現可能です。

また，テストする信号線は信号の名前をキーボードから入力することでも指定可能ですが，先ほどの回路図エディタを呼び出してマウスでクリックし指定することも可能です。

■プログラムのフィッティング

これは，中・大規模なFPGA/CPLD特有の作業です。回路規模が大きくなると機能の実現のしかたにもいろいろな可能性が出てきます。ある機能単位をセルアレイのまとまったところに配置すれば，配線の長さが短くなりますので，高速な動作が期待できます。しかし「はぎれ」が大量に生じてしまいますから，デバイスの持っているセルの使用効率は極端に落ちてしまいます。このようなトレードオフを「ルーティング」や「フィッティング」と呼ばれる過程で検討します。

また，機能を実現するゲートと入出力ピンが離

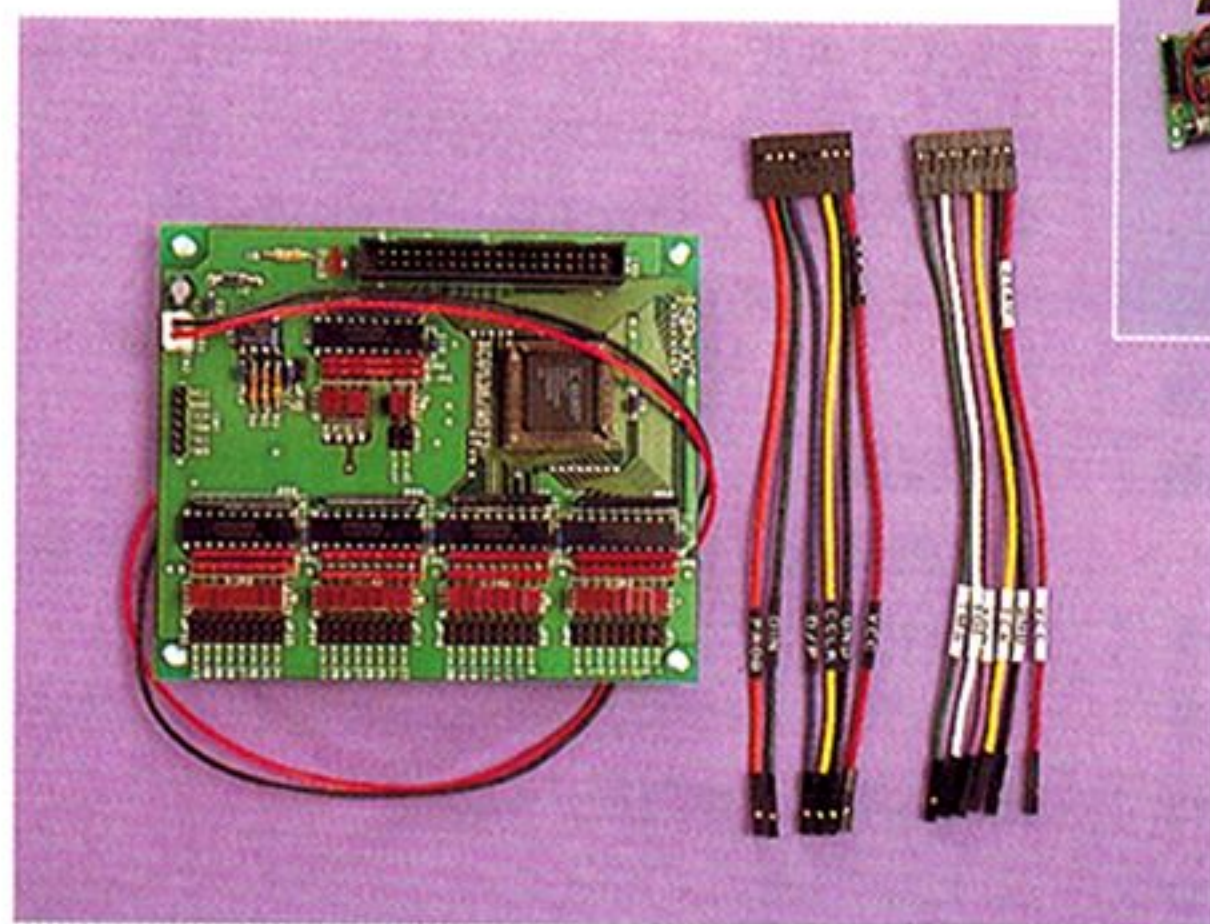



図1　キットの内容物

図2　ISP-XC基板の上面写真

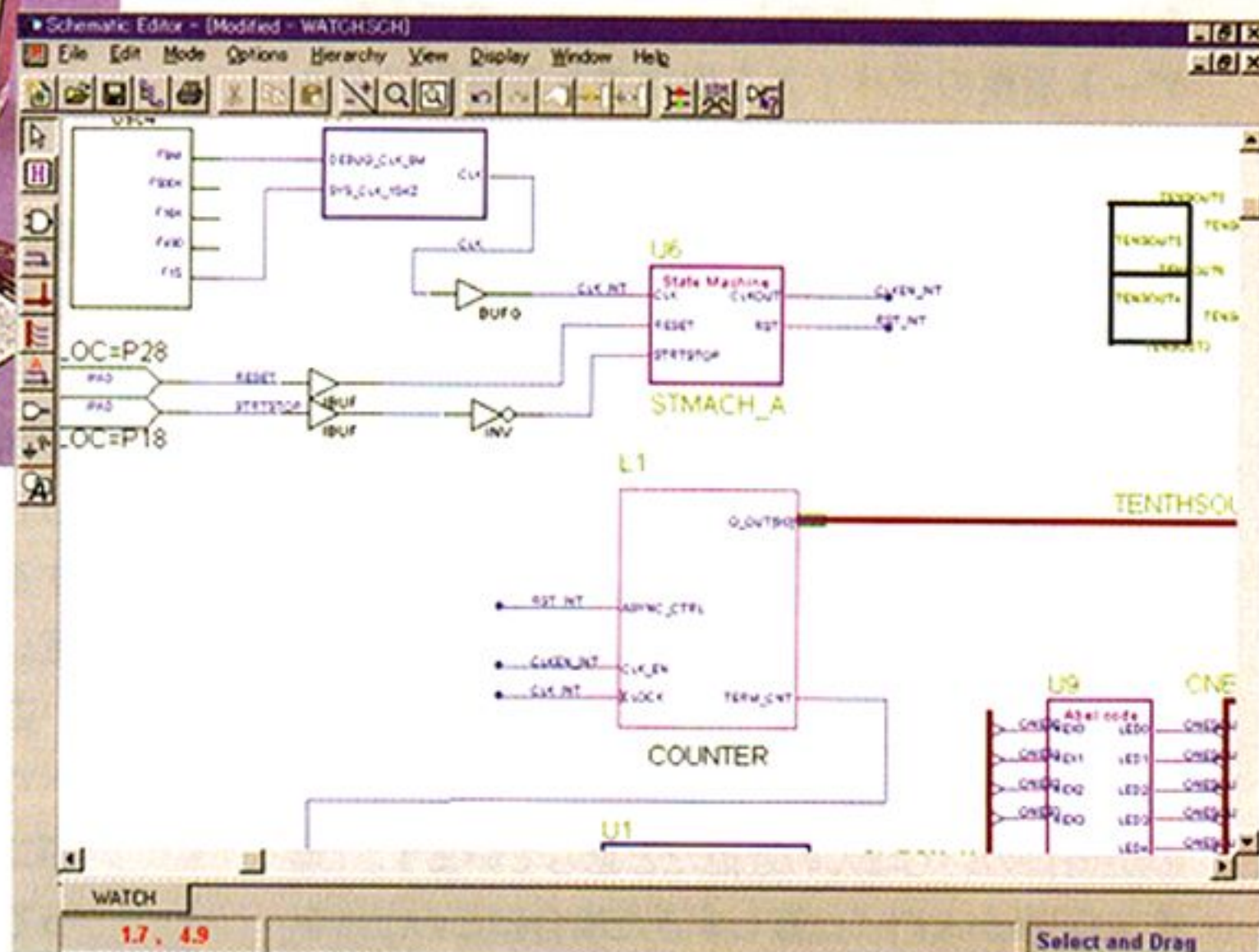



図3　回路図入力

れているとその分性能が落ちます。なるべく「フィッティング」で内部セルの配置・内部配線が決まったあとにデバイスのピン配置を決定したほうが同じデバイスでもより高い性能を発揮できます。とはいっても，プリント基板にはプリント基板の都合というものがありますから，ここでもFPGAのピン配置とプリント基板の配線とのトレードオフを検討します。

小規模なPLDでは内部のルーティングに何種類もやり方があるわけではないので，この工程は必要ありません。

■プログラムの焼き込み

以上のように作成されたプログラムは，付属のJTAGケーブルから実際のデバイスにダウンロード可能です。このJTAGケーブルはPCのプリンタポートに繋げるようになっていて特別なボードは必要ありません。接続先はLPT1かLPT2を選べるようになっていますから「通常使うプリンタ」とは別なものを指定したほうがよいでしょう。間違って電源の入ったFPGAに「ワード」の原稿をプリントアウトしてしまうと危険です。*1

(1)PC99準拠マザーボードで，プリンタポートがひとつしかなくって，ISAスロットがなかったら，いったいどうするのでしょう？　余談ですが，HDLコンパイラにはプリンタポートにセキュリティ用の小道具を差さねばならないものもあったりします。どなたか，PCI接続のプリンタポート拡張基板を販売しているところがあったら教えてください。

■開発ツールそのものについて

私は，エントリーモデルのFDN-BAS-PC-Jを使用しているのですが，このモデルは，

● 使用ゲート数10000以下
● 以下の方法で開発可能（混合可能）
・回路図入力
・状態遷移図入力
・HDLはABELのみ

といった機能をサポートしています。

なお，このDS-FDN-BAS-PC-J以上の機能が必要になったときには，それぞれ，

● VDHL / Verilog-HDL使用可
● VDHL / Verilog-HDL使用可。ゲート数制限なし

というバージョンもあるので，大規模回路にFPGAを使用する場合でも，入門キットで得たノウハウや経験が無駄になりません（ただし，ツールの価格は指数関数的に上昇します）。

このキットは，開発ツールだけでなく，

● GUIベースの統合環境（ラウンチャが少し高級になった感じ。Cコンパイラの統合環境ほど統合されていないが，FPGAの開発はいろいろなソフトが複雑に絡んでくるので，便利）
● GUIベースのインタラクティブなシミュレータ
● 作ったプログラムをデバイスにダウンロードするためのJTAGインタフェイス＆ケーブル（IBM PC機のプリンタケーブルに接続します）
● チュートリアル

が含まれています。これさえあれば，基板を作った日から開発が始められるのですが，基板を作るのさえ面倒だという人のために，秋葉原の若松電子通商(http://www.wakamatsu.co.jp/，電話03（3251）4121）では，XC9536を使った基板とセット販売（26,000円）しているので，私のようなものぐさには最適です。

実は，あまり他社のFPGAは扱ったことがないので，とりあえずXilinxのFPGAツールを紹介しましたが，そのほかにも開発ツールはいろいろあります。高価なサードパーティ製のものしか選択肢がなかった時代もありましたが，FPGA/CPLDメーカー自身が販売促進的な意味あいで，かなり安価に開発ツールの販売を始めたので，いろいろ試してみると面白いかもしれません。

たとえば，Lattice社（http://www.latticesemi.com/）やAltera社（http://www.altera.com）のようにホームページでダウンロード可能な開発ツールを公開している場合もあります。

たいてい，これらの販促ツール（？）ではABELというHDLが採用されていますが，HDLはVHDLかVerilog-HDLに統一しようという業界の流れがあります。しかし，これらの言語を安価・無料で使えるツールというのはまだないようです。

XilinxのVHDL/Verilog-HDLコンパイラも約7万～8万円と値が張りますが，同社のホームページ(http://www.xilinx.co.jp/)でWeb Fitterと呼ばれるXC9500シリーズデバイス限定のオンラインコンパイルサービスをやっているので，こちらも面白そうです。

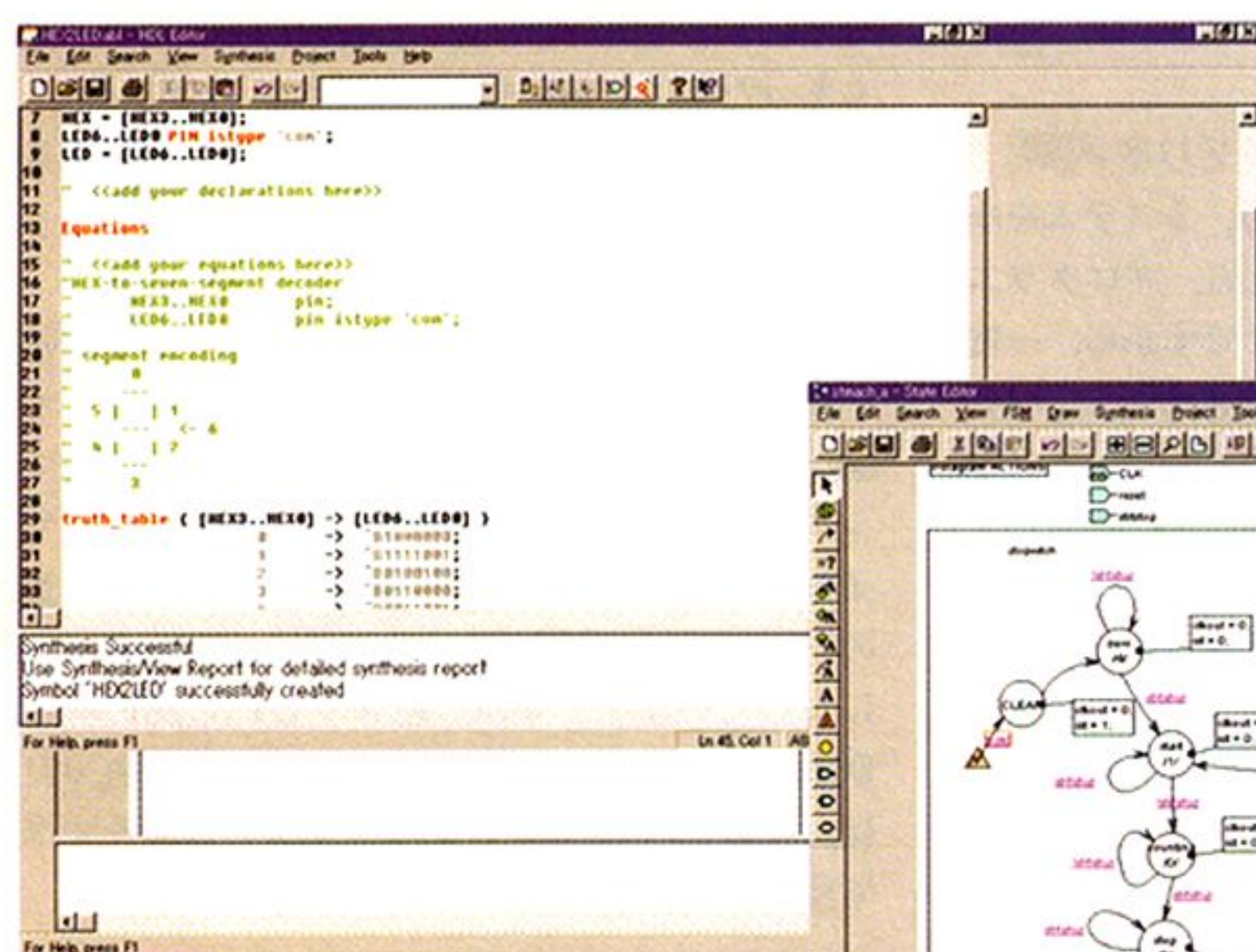



図4　ABEL入力

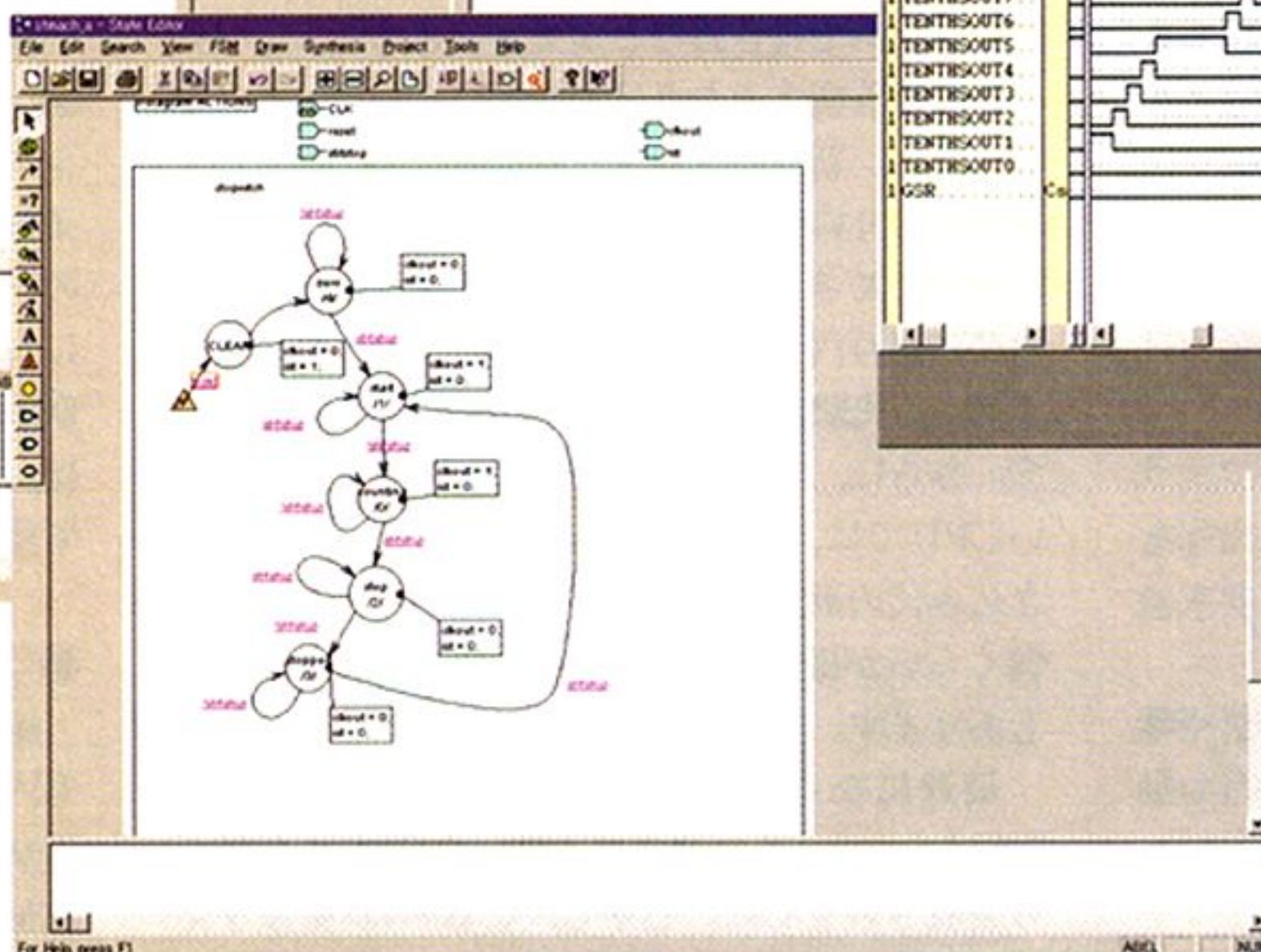



図5　状態遷移図

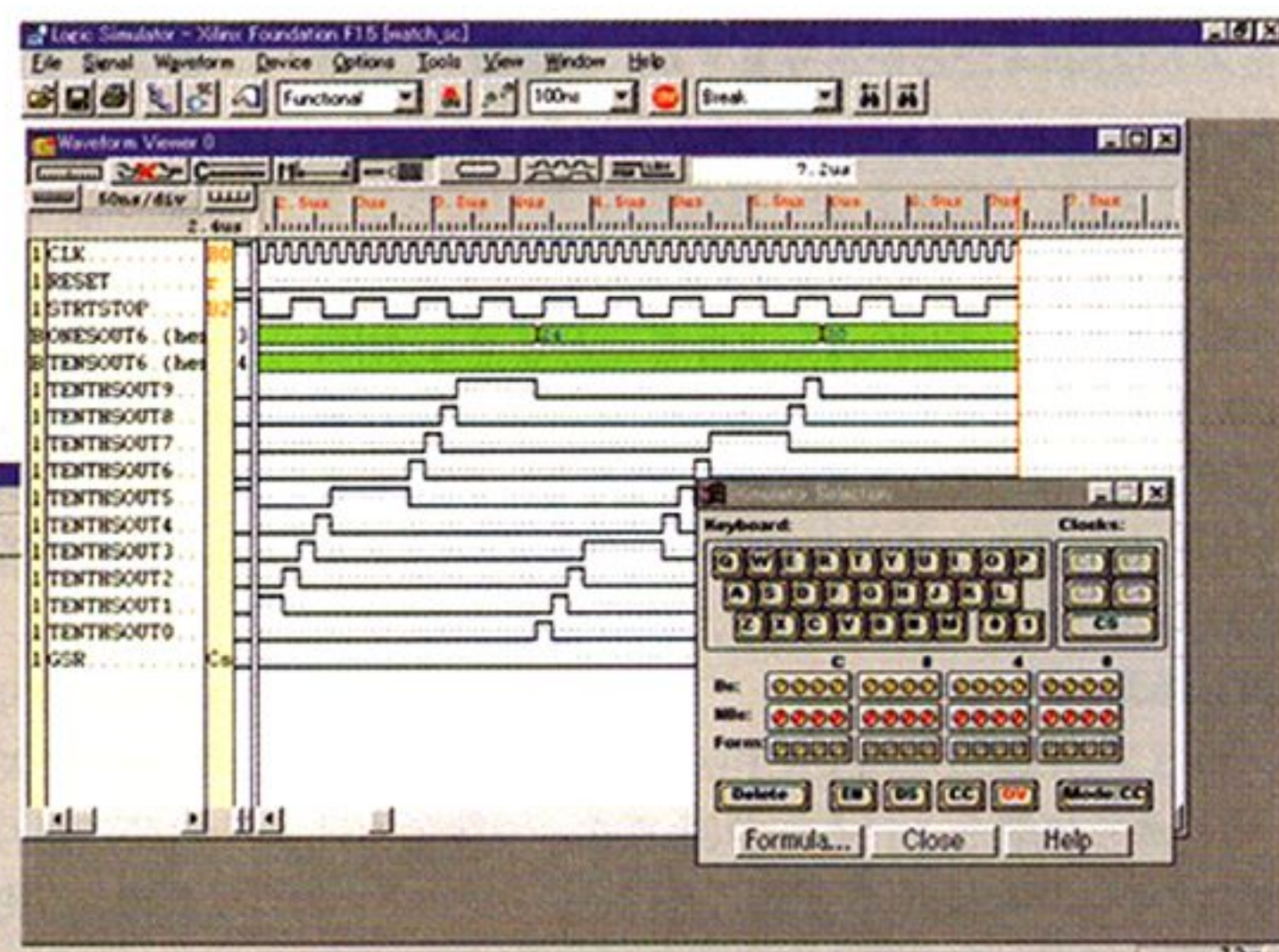



図6　シミュレーション

第2章 PICマイコンとは

高尾　克彦 Takao Katsuhiko

ここでは限りなくプログラミングデバイスに近いCPUであるPICマイコンを紹介する。プログラムによって機能を書き換えられるので，アイデア次第で非常に多才な活用が可能なデバイスだ。今後のPC工作で多用されることになると思われるので，だいたいどんなものかを把握しておいてほしい。

「日本にはね，炊飯器の中にコンピュータが入っていて，帰宅時間になると炊き始め，そのまま温度を保ってくれるのだってあるのよ」と言いながら，これはコンピュータではない，単なるスイッチだ，と思ったが，ジュディーは，「さすが日本ね〜」と，感心している。これで興味を持ち，そろえましょう，ということになった。（カズコ・ホーキ「ロンドンの床下」）

私の理解だと，コンピュータというのは，

- その対応はプログラムで定義されてて，
- 入力に応じて，出力が変わり，
- 一時記憶にメモリを使う

みたいな感じで，このうちひとつでも欠けるとスイッチということになると思うのですが，皆さんいかがでしょう。

コンピュータというと，Pentiumシリーズとか，K6シリーズとか，PCの心臓部が注目されていますが，世の中のコンピュータがすべて300MHzや500MHzで動いていて，2Wも5Wも電力を喰うものばかりではありません。もちろん，x86アーキテクチャ以外にも，MacintoshにはPower PCが使われているぞ，とか，ドリームキャストはSH-4だとかありますが，高速度·大消費電力という点ではx86の親戚みたいなものです。

たとえば，100gもない携帯電話の中にもコンピュータが入っていますが(というと日本語では不自然なので，以下CPUと呼びましょう)，明らかにPentiumなどとは別ものです。だって，放熱板のないPentiumを握りしめたら，火傷しますからね。あんな小さな電池で動作時間が百何十時間もありますから，これは，設計思想からして，別もののCPUです。

おおざっぱにいって，CPUには，

- とにかく性能を上げるもの(大きさ，消費電力は問わない。Pentiumなどがこれ)
- とにかくシステムを小さくするもの(ペリフェラルをとりあえず突っ込むSystem On Chip指向なものや，プログラムの密度を上げてメモリ要求量を下げようと，まじめに考えたものなど)
- とにかく消費電力を小さくするもの(腕時計や電子手帳なんかは限られた小さな電池で何カ月も動作しなければならないので，それはそれはたいへんな苦労のようです)

などの種類があり，それぞれに異なる設計思想のCPUが開発されています。もちろん，ひとつの目標に特化するだけでなく，「過去の資産を継承しつつ，性能をそこそこ上げて，多くの自社製特許を投入する」というバランスの上に成り立っているものもあります。

今回，紹介するPICマイコンはPentiumやPowerPCとは違い，これ以上簡素化したらもはやCPUではなくなってしまうのではないかというくらい，ギリギリまで簡素化したCPUです。

■PICマイコンとは？

Microchip Technologyは，1989年にGeneral Instrumentのメモリ部門が分離，独立して設立された会社です。今回紹介する小型組み込みマイコンのほかにも，EEPROMなどを扱っています。詳しい話は同社のホームページ(http://www.microchip.com/)を見ていただくとして，主力製品のひとつであるPICマイコンの特徴を見ていきましょう。

まず，デバイスそのものの特徴ではありませんが，アマチュアから見た場合，以下のような特徴があります。

●プログラム用フラッシュメモリを内蔵

一般に，組み込みマイコンでは，システムを小型化し低消費電力を実現するため，プログラムROMを内蔵しています。ROMですから，一度プログラムを書き込むと変更がききません。プログラムを改変するときは基板から古いデバイスを取り外し，新しいプログラムを書き込んだデバイスを代わりに実装します。たいてい古いデバイスは再利用できませんので，プログラムを改変するたびに何百円もするデバイスを捨てなければなりません。実験は，常に財布と相談しながら行います。さらに，マスクROM*1しかサポートしてないCPUでは，プログラムにバグが見つかると書き込み代の数十万円と1ロット分のデバイス(100個くらいが標準か？)がパーになってしまう場合もあります。

最近になって，このプログラム用メモリにRAMやフラッシュメモリを使ったCPUが各社から発表されました。しかし，これらは(量産予定のある)大口顧客が開発用に使うことを想定していたのでとても高価な場合が多々ありました。

今回，紹介するPIC16F84は1K Wordと小容量ながら1000回以上更新ができるプログラム用フラッシュメモリを内蔵しつつ，400〜500円で買えるという，我々アマチュアユーザーにとっては，大変ありがたいデバイスです。

もちろん，Microchip Technologyは，PIC16F84で開発したプログラムをマスクROM版のデバイスに書き込んで出荷するサービスも行っています。私はやったことがないので，これ以上はなにもいえないのですが，興味がある方は同社のホームページ上で紹介されている代理店に問い合わせてみるとよいかもしれません。

●扱いやすいパッケージ

組み込み用のマイコンはとても小型化指向が強いため，パッケージもどんどん小さくなってきて，ピンとピンの間が0.5ミリか0.65mmピッチくらいが標準となってきています。あまりピンとこないかもしれませんが，

||||||||||(←これが0.5mmピッチです)

の1本1本をショートさせずにハンダづけをするには，高価な機械か高度な技能が必要です。最近ではピンすら出していない直付け用のパッケージもあったりしてアマチュアにはお手上げです。

PICシリーズは，ちゃんとアマチュア向けにも(？)昔ながらの2.54mmピッチのデバイスを出しています。

ピッチサイズだけの問題なら昔のマイコンを使い続けるということもできますが，往々にして36pinとか40pinとかパッケージサイズが巨大でシステム全体も大きくなってしまいます。PICマイコンではピンの種類も厳選され，少ないピン数のパッケージでもちゃんとシステムが組めるようになっています。今回，紹介するPIC16F84はDIP18ピンですが，同じファミリのPIC12C671は，なんとDIP8ピンに必要最低限の機能をすべて集約しています。

●安価な開発ツール

組み込みマイコンの開発ツールはだいたい数十万円くらいするものですが，PICマイコンの開発用ソフト(アセンブラ＋シミュレータ＋統合環境)はMicrochip Technologyのホームページからダウンロード可能で，ハードもだいたい三万円台

表1 PICマイコンのファミリ

型名	プログラムメモリ	データ用RAM（バイト）	データ用EEPROM（バイト）	最大動作クロック（MHz）	I/Oポート	A/Dコンバータ	シリアルポート	DACコンバータ	コンパレータ	タイマ	外部ピン割り込み	電源電圧範囲	パッケージ
8ピン・パッケージ													
PIC12C508	512 x 12	25		4	6					1+WDT		2.5-5.5	8P, 8SM
PIC12C509	1024 x 12	41		4	6					1+WDT		2.5-5.5	8P, 8SM
PIC12C670	512 x 14	80		4	6	8-bit x 2 ch				1+WDT		2.5-5.5	8P, 8SM
PIC12C671	1024 x 14	80		4	6	8-bit x 2 ch				1+WDT		2.5-5.5	8P, 8SM
PIC12F680	512 x 14（Flash）	54	16	4	6					1+WDT		2.5-5.5	8P, 8SM
PIC12F681	1024 x 14（Flash）	54	16	4	6					1+WDT		2.5-5.5	8P, 8SM
12-bit幅命令アーキテクチャ（命令数=33）													
PIC16C52	354 x 12	25		4	12					1		3.0-5.5	18P, 18SO
PIC16C54	512 x 12	25		20	12					1+WDT		3.0-5.5	18P, 18JW, 18SO, 20SS
PIC16C54A	512 x 12	25		20	12					1+WDT		2.5-6.25	18P, 18JW, 18SO, 20SS
PIC16C55	512 x 12	24		20	20					1+WDT		2.5-6.25	28P, 28JW, 28SP, 28SO, 28SS
PIC16C56	1024 x 12	25		20	12					1+WDT		2.5-6.25	18P, 18JW, 18SO, 20SS
PIC16C57	2048 x12	72		20	20					1+WDT		2.5-6.25	28P, 28JW, 28SP, 28SO, 28SS
PIC16C58A	2048 x12	73		20	12					1+WDT		2.5-6.25	18P, 18, 18SO, 20SS
14-bit幅命令アーキテクチャ（命令数=35）													
PIC14C000	4096 x 14	192		20	20	8 SLAC	2C / SMB	8-bit x 2ch		2+WDT			28SP, 28SO. 28SS, 28JW
PIC16C554	512 x 14	80		20	13					1+WDT	Yes	2.5-5.5	18P, 18JW, 18SO, 20SS
PIC16C556	1024 x 14	80		20	13					1+WDT	Yes	2.5-5.5	18P, 18JW, 18SO, 20SS
PIC16C558	2048 x 14	128		20	13					1+WDT	Yes	2.5-5.5	18P, 18JW, 18SO, 20SS
PIC16C61	1024 x 14	36		20	13		I2C / SPI	8-bit x 1ch		3+WDT	Yes	3.0-6.0	18P, 18JW, 18SO
PIC16C62	2048 x 14	128		20	22		I2C / SPI	8-bit x 1ch		3+WDT	Yes	3.0-6.0	28SP,28SO, 28SS, 28JW
PIC16C62A	2048 x 14	128		20	22		USART / I2C / SPI	8-bit x 2ch		3+WDT	Yes	3.0-6.0	28SP,28SO, 28SS, 28JW
PIC16C63	4096 x 14	192		20	22		I2C / SPI	8-bit x 1ch		3+WDT	Yes	3.0-6.0	28SP, 28SO, 28JW
PIC16C64	2048 x 14	128		20	33		I2C / SPI	8-bit x 1ch		3+WDT	Yes	3.0-6.0	40P, 40JW, 44L, 44PQ
PIC16C64A	2048 x 14	128		20	33		USART / I2C / SPI	8-bit x 2ch		3+WDT	Yes	2.5-6.0	40P, 40JW, 40L ,44PQ, 44PT
PIC16C65	4096 x 14	192		20	33		USART / I2C / SPI	8-bit x 2ch		3+WDT	Yes	2.5-6.0	40P, 40JW, 44L, 44PQ,
PIC16C65A	4096 x 14	192		20	33					3+WDT	Yes	2.5-6.0	40P, 40JW, 44L, 44PQ, 44PT
PIC16C620	512 x 14	80		20	13					1+WDT	Yes	3.0-6.0	18P, 18S0, 18JW, 20SS
PIC16C621	1024 x 14	80		20	13					1+WDT	Yes	3.0-6.0	18P, 18S0, 18JW, 20SS
PIC16C622	2048 x 14	128		20	13					1+WDT	Yes	3.0-6.0	18P, 18S0, 18JW, 20SS
PIC16C642	4096 x 14	176		20	22					1+WDT	Yes		28SP,28SQ, 28JW
PIC16C662	4096 x 14	176		20	22					1+WDT	Yes		40P, 40JW, 44L, 44PQ, 44PT
PIC16C710	512 x 14	36		20	13					1+WDT	Yes	2.5-6.0	18P, 18S0, 18JW, 20SS
PIC16C71	1024 x 14	36		20	13	8bit x 4ch			2	1+WDT	Yes		18P, 18S0, 18JW,
PIC16C711	1024 x 14	68		20	13	8bit x 4ch			2	1+WDT	Yes	2.5-6.0	18P, 18S0, 18JW, 20SS
PIC16C72	2048 x 14	128		20	22	8bit x 4ch			2	3+WDT	Yes	2.5-6.0	28SP,28SQ, 28JW, 28SS
PIC16C73	4096 x 14	192		20	22	8bit x 5ch	I2C / SPI	1 PWM	2	3+WDT	Yes	2.5-6.0	28SP,28SQ, 28JW
PIC16C73A	4096 x 14	192		20	22	8bit x 5ch	USART / I2C / SPI	2 PWM	2	3+WDT	Yes	2.5-6.0	28SP,28SQ, 28JW
PIC16C74	4096 x 14	192		20	33	8bit x 5ch	USART / I2C / SPI	2 PWM		3+WDT	Yes	2.5-6.0	40P, 40JW, 44L, 44PQ,
PIC16C74A	4096 x 14	192		20	33	8bit x 8ch	USART / I2C / SPI	2 PWM		3+WDT	Yes		40P, 40JW, 44L, 44PQ, 44PT
PIC16F83	512 x 14（Flash）	36	64	10	13	8bit x 8ch	USART / I2C / SPI	2 PWM		1+WDT	Yes	2.0-5.5	18P, 18SO
PIC16C84	1024 x 14（EEPROM）	36	64	10	13					1+WDT	Yes		18P, 18SO
PIC16F84	1024 x 14	68	64	10	13					1+WDT	Yes	2.0-5.5	18P, 18SO
PIC16C923	4096 x 14	176		8	52		I2C / SPI			3+WDT	Yes		64SP, 64CL, 68L, 64PQ
PIC16C924	4096 x 14	176		8	52	8bit x 5ch	I2C / SPI			3+WDT	Yes		64SP, 64CL, 68L, 64PQ
16-bit 幅命令アーキテクチャ （命令数=55, 8x8乗算機付き）													
PIC17C42A	2048 x 16	232		33	33		USART			4+WDT	Yes		40P, 40JW, 44L, 44PQ, 44PT
PIC17C43	4096 x 16	454		33	33		USART			4+WDT	Yes		40P, 40JW, 44L, 44PQ, 44PT
PIC17C44	8192 x 16	454		33	33		USART			4+WDT	Yes		40P, 40JW, 44L, 44PQ, 44PT
PIC17C756	16384 x 16	902		33	50	10-bit x 12ch	USART / I2C / SPI			4+WDT	Yes		64SP, 64C+, 68L, 64PQ

パッケージ

CL: 紫外線除去用窓付きPLCC　L: PLCC　PQ: PQFT　SP: DIP
JW: 紫外線除去用窓付きDIP　P: DIP　SO: SOIC　SS: SSOP　PT: TQFP

と組み込み用途のツールとしては破格に安く提供されています。

この開発ツールのハード部分というのは必ずしも必要ではなく，開発ソフト＋プログラムライタでも十分実用に耐えます。このプログラムライタと書き込みソフトも非常に安価に作る方法が同社のホームページにアプリケーションノートとして紹介されています。がんばれば1000円以下でできるのではないかと思います(図1)。

●豊富なアプリケーションノートと秋月キット

皆さんがZ-80や68000のマシン語を使い始めたときのことを思い出してください。データシートや解説書だけでなく，他人の作ったプログラムを見たり改造したりしたことがとても理解に役立ったはずです。

組み込みマイコンも同様でいくら立派なデータシートがあっても，すぐにプログラムが書き始められるわけではありません。Microchip Technologyでは，ホームページに表1のように多くの参考プログラムをアプリケーションノートという形で掲載しています。

また，秋葉原にある秋月電子通商というお店はPICマイコンを使ったさまざまなキットを開発・販売していますが，そのほとんどがソースリスト付きです。このお店は独自のアセンブラを使用しており，Microchip Technologyのアプリケーションノートよりもプログラムを理解するのが簡単になっています。キット自身もユーザーが独自に改造することを考慮し，ボードスペースに余裕を持たせています。

*1 ユーザーが自分でプログラムを書き込むことができず，半導体工場にお願いする。

■その他の特徴

組み込みマイコンは数百種類以上もあります。すべてに目を通しているわけではないのであくまでも私見ですが，Microchip TechnologyのPICマイコンはほかの組み込み用マイコンと比較して以下のような特徴を持っています。

●RISC型プロセッサ

本誌春号の中森氏の記事で紹介されたように，RISCというとPower PCやR3000などが有名です。これらがRISCアーキテクチャを採用したのは，

- 回路を簡素化し，動作速度を上げる
- パイプライン化により並列動作を行う

などの理由によりますが，PICマイコンの場合は別の理由によります。

RISC(Reduced Instruction Set Computer)という言葉どおりに回路構成をシンプルにするのは同じですが，シリコンの小型化(低価格化)，低消費電力を実現するために使われています。

このためハードウェア的に見て複雑な動作を行う命令(剰余算など)は実装されてませんし，CISC型マイコンによくあるブロック転送命令やストリング命令などもありません。

また，PICマイコンのパイプライン化はそんなに難しくないと思われますが，おそらくは無駄に消費電力を上げないという観点から見送られています。PICマイコンでは，1動作サイクルに4クロック(命令フェッチ，デコード，実行，書き込み)必要です。ですから，カタログにある最高動作速度20MHzというのは，1秒間に5000000(＝20M/4)命令が実行可能だ，という意味です。

●ハーバードアーキテクチャ

組み込みマイコンでは珍しく，PICマイコンではハーバードアーキテクチャが採用されています。

このハーバードアーキテクチャとは，プログラムとデータが違うメモリ空間に配置されたプロセッサの構成です。ですから動作中に間違ってプログラムを書き換えてしまうとか，ワークエリアのデータを命令と解釈し実行してしまうといった事故はなくなります(その代わり自己書き換えプログラムは使えません)。そのほか，プログラム空間からコードをフェッチする回路とデータ空間からデータを読み込む回路が並列動作しやすいというメリットがあります。たとえば，Z80で，

```
LD   A, H
```

は1サイクル(＝4クロック)で実行されますが，

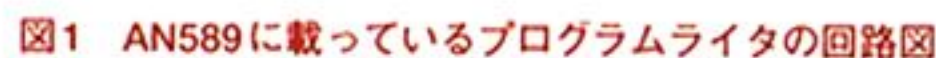
```
LD   A, (HL)
```

は，2サイクル(＝7クロック)必要でした。これはZ80が同じメモリ空間をプログラムとデータ用に使用しているため，プログラムのフェッチが終わらないとデータを読み出せないためです。

PICマイコンはハーバードアーキテクチャを採用していますので，RAMへのデータ書き込み/読み出し命令は，1サイクル(＝4クロック)で行われます。

また，2つのメモリ空間が分かれているため，アドレス情報が少なくてすむというメリットもあります。たとえば，コード230Word＋データ50Wordのプログラムがあった場合，

(1)非ハーバードアーキテクチャ
　230＋50＝280で9ビットが必要
(2)ハーバードアーキテクチャ
　コード用に8ビット，データ用に6ビットが必要
　必要ビット数が増えて物理的に信号線が1本増

図1　AN589に載っているプログラムライタの回路図

えるのはさほど問題ないのですが，命令のエンコード幅から引数に何ビット使えるのかを決める際には必要ビット数は重要になります。たとえば，PIC16F84の場合，命令のエンコード幅は14ビットですが，このうち引数に8ビット使用しています。残った5ビットで命令の種類を規定します。5ビット残っていれば32種類の命令が定義できますが，引数に9ビット使ってしまうと4ビットしか残っていないので16種類の命令しか定義することができません。

■PICマイコンファミリ

現在のところ，PICマイコンファミリには，表1のような種類があります。

まず，CPUのコアとして，

PIC16c5x：
　　インストラクション長12ビット（命令数33）
PIC16xxx：
　　インストラクション長14ビット（命令数35）
PIC17xxx：
　　インストラクション長16ビット（命令数55）

の3種類があります。それぞれのコアでインストラクション長が違いますが，それは命令の引数に扱える値が異なってくるだけで，命令の種類などはそんなに変わりません。PIC16c5xのプログラミング経験者がPIC16xxxでプログラミングを行う際にもそんなにとまどうことはないでしょう。

組み込みマイコンの常として，CPUコア単体ではなく，実際のデバイスはCPUコア＋ペリフェラルという構成になっています（シリアルI/Oやタイマなど）。

表1にあるようにペリフェラルの組み合わせによって，かなりの数のファミリがありますので扱う用途に適したデバイスを探してください。実際のところ，シリアルIOはパラレルIOの何本かのピンでも再現できますし（参考文献1参照），アナログコンパレータも精度を気にしなければ，パラレルIOとちょっとした外付け部品で組むことができますので，我々アマチュアの場合，入手性を考えてとりあえずPIC16F84を検討し，どうしてもだめだった場合のみ，ほかのファミリを考えるということになると思います。

■プログラム開発に必要なもの

パソコンでプログラムを開発する場合は，プログラミング環境と動作環境が同じマシンでしたが，組み込みマイコンは最新のプログラミング環境をサポートするほど強力ではないので，普通，プログラミングはパソコン（あるいはワークステーション）で行い，できたプログラムを組み込みマイコンで実行します。

●パソコン

プログラムを作ったり，検証したりするのに使います。ハードウェア開発の常としてMacintosh系はサポートされていないので，IBM-PC互換機が必要です。ただし，アセンブラ/シミュレータはWindows上で動くので，対応するプログラムライタが手に入れば，PC-98xxでも開発は可能です。アセンブラ/シミュレータともにたいしたことをやっているわけではないので，エディタが許せばひと昔前のPCでも問題ありません。

●アセンブラ

純正品として，Microchip Technologyのホームページ（http://www.microchip.com/）からMPLABという統合型開発ツールがダウンロードできます。これはエディタ・アセンブラ・リンカを含めた開発環境というだけでなく，別売りのCコンパイラやICEもそこから操作できるという，非常に高機能なものとなっています。ていねいなチュートリアルもついていますので入門にも適しています。

そのほかにも，秋月電子通商の"AKI-PICプログラマーキット"（7000円，40 pin ZIFソケットなしで，5700円）に含まれるアセンブラがあり，これがなかなか秀逸です。Microchip Technology純正のニーモニックに加えて独自のマクロ命令が含まれており，大変使いやすくなっています。

第3の方法としては，今号のCD-ROMに含まれるアセンブラがあります。これもMicrochip Technology独自のニーモニックに加えて独自のニーモニックが使用できるようになっています。

以上，3つの中からお好きなものを選んでください。

●プログラムライタ（Program Writer）

せっかく作ったプログラムを，PC上でシミュレートしているだけでは面白くありません。実際のデバイスに書き込んで動作させるには，プログラムライタと呼ばれる装置が必要です。Microchip Technologyの純正開発環境にもプログラムライタが含まれていますので，それを使用してもいいですが，せっかくアセンブラ・シミュレータなどを無料で手に入れられるのですから，プログラムライタも安く済ませたい人は以下のような選択肢があります。

●Microchip Technologyのホームページへ行き，Application Note 589（PC Based Development Programmer for the PIC16C84）をダウンロードする

そこには，図1に示すような簡潔な回路と書き込みプログラムが紹介されています。これはPCのプリンタポートから動作を制御できるようになっています。タイトルからはPIC16C84専用みたいな印象を受けますが，PIC16F84でも使用可能です。一部，日本では馴染みのない部品が使用されていますが，さほど特性が重要なわけでもありませんので，それぞれ，

3端子レギュレータ
　LM340-5 → 7805
ダイオード
　1N4148 → 1S1588
PNPトランジスタ
　2N3906 → 2SA1015
NPNトランジスタ
　2N3904 → 2SC1815

のように代用できます。また，PC-98xxにはないACK端子を利用してプログラムライタからの信号をPCへ送っていますが，ソフトウェアが対応すればBUSY端子でも問題ありません（参考文献2参照）。

●秋月電子通商の「AKI-PIC　プログラマーキット」を購入する

前述のように，これは単なるプログラムライタではなく，独自のアセンブラやシミュレータ（DOSベース。純正ニーモニックのみサポート）が付属して，7000円（40 pin ZIFソケットなしで，5700円）という低価格で販売されています。図1の回路がいくらシンプルでも，部品を集める手間，配線する手間などを考えると，この選択がいちばん安くつくのではないでしょうか。プリンタポートを利用すると電気回路はシンプルになりますが，コネクタのピッチが基板と異なるので実装が大変な場合があります。キットならば，この部分はハンダづけすればよいようになっているのでラクチンです。

●サードパーティ製のプログラムライタを使う

私はこの方法を使いました。昔，X68000用の68030アクセラレータを開発した際に購入したGALライタが，ソフトウェアのアップグレードだけで，PICマイコンにも対応できました。そのほかにも，ひととおりのROM/PLD/GALに対応しているだけでなく，数百種類の組込用CPUにも対応しています。

確かに，各デバイスごとに安価なプログラムライタを自作していったほうが安いのかもしれませんが，5年後に動かなくなっていたり，やたらと不安定だったりすると，もう一度作り直さなければなりません。私の製作した回路は大事に保存しておいても壊れてしまう場合が多々ありました。また，自作回路やキットは書き込みの回路が簡素化されているため，新ファミリが出た場合にまたライタを自作しなければならない可能性もあります。

私が使ったプログラムライタはHi-Lo SystemsのAll-07というものですが各ピンのタイミング・電圧がとても柔軟に設計されているらしく，ソフトウェアの対応次第でほとんどのデバイスに対応するといっています（今回のPIC16F84も含めて）。そのアップグレード用のソフトウェアも同社のホームページに公開されています（http://www.hilosystems.co.tw/）。

書き込み機とDIP用アダプターをあわせると10万円くらいしてしまいますが，これからハードウェア工作を行う人ならすぐに元はとれると思います。

Boxed Article

PICマイコンの情報入手法

一応，「知っている人は知っている」というレベルまで到達したPICマイコンですが，Oh!Xの読者のなかには今回初めて目にする方もいらっしゃるかもしれません。

デバイスの限界が定性的に計れるレベルにあるにもかかわらず，次々と，アマチュアユーザーによって驚くような使い方が示されるという，Oh!Xの読者にとってはたいへんエキサイティングな分野だとは思うのですが，Oh!Xのメインテーマにするにはちょっと無理があるかもしれません。今回の記事のほかにも，さまざまな情報入手方法がありますので，今回，興味を持たれましたら，以下の情報にもアクセスしてみてください。

●雑誌

月刊「トランジスタ技術」(CQ出版)

私が知る限り，もっとも歴史が長く，ページが多く，販売量が多いハードウェア系の雑誌です。以前はメーカーの技術者をメインターゲットにしていたみたいですが，最近では記事のレベルを少し下げてアマチュアエンジニアも参加できるようになってきました。

PICマイコンへの取り組みも早く，1995年にはすでに「ワンチップマイコンで行こう」と題して特集項目にもなっていました。

以下に，これは凄いと私が思った過去の記事を列挙しておきますので，図書館などで見つけてください。同誌ではバックナンバー記事のコピーサービスも行っているようです。詳しくは同誌最新号で確認してください。Oh!Xでこんなに宣伝しちゃっていいのかなぁ（編注：問題ありません）。

- 渡辺 明禎「PS/2使用ジョイパッドの製作」1995年12月号p291 - 297

PIC16C84で，スーパーファミコンのジョイパッドをPS/2仕様のマウスとして使う試みです。

- 幾島 康夫「学習型バーサライタの製作」1995年12月号p298 - 397

バーサライタとは，暗闇で真横に動かすと残像として文字が浮き上がってくる不思議な機械です。そのアイデア自体目新しいものではありませんが，その文字列の入力方法に，この製作記事ではPCのディスプレイを使用しています。CRTのある3点を，それぞれDATA,CLOCK，SYNCとして，CdSで拾い，PIC16C84のメモリに格納します。

- 瀬戸口 豊「ワンチップ・ブレークアウト・ゲームの製作」1997年5月p353 - 360

これは凄いです。PIC16C84で「ブロック崩し」を実現しています。しかも，出力はビデオ信号です。さすがにPIC16C84ではVRAMを持つわけにはいかず，各オブジェクトの座標判定で白/黒の表示をしているようですが，それでも，まさかPICマイコンでビデオ表示が行えるとは思いませんでした。

そのほかにも，A/Dコンバータを持たないPIC16C84でパドル（つまみ）信号の処理法などが紹介されています。

- 後閑 哲也「超音波距離計の製作」1999年5月号p208 - 216

PIC16F84で，40KHzの超音波を射出し，その反射波が帰ってくるまでの時間から，対象物との距離を測定します。

●書籍

Myke, Predko "Programming and customizing the PIC Microcontroller", McGraw Hill, 1998

ざっと見渡したところ，日本語の書籍でPICマイコンを扱ったものはまだないようです（厳密にいうと，復活Oh!Xは雑誌ではないから，ひょっとしてこの記事が日本初の「書籍」になるのかな？）。英語になってしまいますが，この本が唯一，私が見つけることのできたPICマイコンの書籍です。

PICマイコンの説明に始まり，PIC16C84（PIC16F84でも使用可）を中心とした42もの実験（LEDの点灯から，最後はPIC16C73Aで，PIC16Cxxのエミュレータを作ってしまうなんていうのも），16ビットの四則演算ライブラリ，各種LCDドライバインタフェイス例と，たいへん内容の濃いものとなっています。プログラムは，付属のFDに収められており，打ち込まずにそのまま使用できます。また，多少込み入った回路には基板のパターン図も載っています（ただし，両面基板です）。

●インターネット

http://www.microchip.com/

Microchipホームページです。データシートから各種アプリケーションまで幅広く，大口ユーザー，アマチュアと分け隔てなく配布しています。純正アセンブラのMPASMや統合型開発環境のMPLABも同社のホームページからダウンロードすることが可能です。PIC16F84などの人気デバイスは最近日本語のデータシートも発行されましたので，ぜひ入手されることをおすすめします。

http://www.mal.co.jp/

（株）マイクロ・アプリケーション・ラボラトリーのホームページです。PICマイコン関連の輸入・独自のキット開発・販売などを手がけているようです。

http://www.myke.com/

先に挙げた書籍の筆者のホームページです。もう，なんというか，8051に始まって，PICマイコン，SVRなど，もろもろ，組み込みマイコンの大家です。各種リンクが役に立ちます。

●秋月電子通商

PICマイコンの日本における草の根運動はここから始まっていっても過言ではないでしょう。数年前からPICマイコンのキットを輸入，あるいは自社企画し，販売しています。企画力・技術力はかなり高く，独自にRS-232Cの信号をLCDに出力するキットや，DTMF解読キット，疑似交換機キットなどを製作しています。特にAKI-PICプログラマーキットは単なるROM焼き機でなく，秋月電子オリジナルのアセンブラとシミュレータを付属したたいへん完成度の高い製品となっています。

また，書籍と違ってキットですから，購入すれば説明書やプログラムとともに部品も揃っており，その日のうちに作業が開始できるのも嬉しい点です。

連絡先
http://www.tomakomai.or.jp/akizuki
Tel ：03(3251)1779
Fax：03(3251)3357
お店：
JR秋葉原駅，電気街口下車，徒歩5分。
Laox The Computer館近く。

参考文献

(1) Microchipアプリケーションノート AN510 "Implementation of an Asynchronous Serial I/O"
(2) 野澤 康夫「パラレル・ポート接続のPIC16C84ライタの製作」トランジスタ技術1995年12月号p248 - 262

第3章 PIC16F84の詳細

高尾　克彦 Takao Katsuhiko

それでは実際にPICチップを使っていくことを考えよう。まず代表的なPICマイコンチップであるPIC16F84の詳細について解説してみたい。多少毛色は違うがアセンブラプログラムの経験者なら，大まかなアーキテクチャと命令セットさえわかれば，なんとかプログラムを作成できるはずだ。

■PIC16F84のアーキテクチャ

ここでは，数あるPICマイコンファミリのなかでも，入手が容易で応用範囲がもっとも広いと思われるPIC16F84に絞って話を進めていきましょう。

PIC16F84は以下のような構成となっています。

●PIC16xxx CPU Core

14ビット幅のインストラクションを35個持っています。PIC16xxxファミリはすべてこのコアを使っているので，ファミリ間でプログラムの互換性があります。どんなCPUコアなのかは後述します。

●プログラム用フラッシュメモリ(14ビット×1K Word)

プログラム格納用のメモリです。PIC16F84は外部メモリにプログラムを置けませんので，実際のプログラムはすべて1K Word以内に収めなければなりません。このPIC16xxx CPUコアは1命令＝1Wordですから，1024ステップということになります。

フラッシュメモリというのは，電気的に消去/再書き込みが可能で電源を切っても内容が消えないという特性を持っています。最近のPCではBIOSの格納などによく使われています。以前のEPROMに比べて，消去用の紫外線が要らない，アクセススピードがやや速いなどの特徴があります。

PIC16F84のフラッシュメモリは，1000回以上の消去/書き込みが保証されていますので，ひとつのデバイスで1000回までデバッグのためのトライ＆エラーが行えるということです。また，フラッシュメモリはRAMと違って電源を切ってもメモリ内容は保存されますから，電源投入後，すぐにプログラムを開始できます。

●EEPROM(8ビット×64。100万回以上の消去/書き込みが可能)

プログラム用のフラッシュメモリが，ライタでの書き込みを想定しているのに対し，このEEPROMはPICマイコンのプログラム中からの書き込み/読み出しを想定しています。RAMに比べてアクセス方法が複雑ですが，電源を切っても内容が保存されるという特徴があります。

また，PICマイコンはプログラムメモリに定数用のデータエリアを確保できないのですが，EEPROMに格納したデータを電源投入時にRAMへ転送し，それをデータとして用いることができます。

●RAM(8ビット×68)

プログラムで用いる一時的な変数などを保存します。また，PICマイコンはI/O空間がないので，タイマやI/Oポートの制御などもこのRAM空間から行います。ペリフェラルだけでなくCPUのレジスタも割り当てられています。

このため，Microchip Technologyのデータシートでは，単なるRAM空間ではなくレジスタファイルと呼ばれていますが，本誌ではPC用のCPU(Z80，68K，x86)に詳しい読者に親しみやすいよう，あえてRAM空間と表記します。

アドレスは，0x00～0x4fまでの80Wordありますが，このペリフェラルなどに0x00～0x0bの12Word予約されているので，実際に，ユーザープログラムの変数領域として使えるのは68Wordのみです。

●タイマ

1/2/4/8/16/32/64/128/256分の1のいずれかを設定できるプリスケーラ(Pre-Scaler)を持った8ビットのタイマが1チャンネルあります。

プリスケーラとは，入力されたクロックが速すぎる場合，タイマにちょうどよいスピードに落とすためのものです。たとえばPIC16F84に動作クロックとして10MHzを与えている場合，プリスケーラで1/256に分周すれば，

10MHz÷256＝39.1kHz

まで落とせます。これでもまだ速すぎる場合は外部回路を組まなくてはなりません。

そのほかにも，

動作中でもタイマの値を読み書き可能

クロック源としてPIC16F84全体のクロックの4分周したものか，外部ピン(3番ピンRA4と共用)を選択可能

割り込み要求源として動作可

という特徴があります。

また，このタイマはウオッチドッグタイマ(Watch Dog Timer)としても使用可能です。

ウオッチドッグタイマとして用いたときの割り込み周期は，約18msecのクロック源に固定され，プリスケーラは1/2/4/8/16/32/64/128分の1のいずれかです。この約18msecのクロック源は，動作温度や電源，製造プロセスによってばらつきがあり，あまり正確ではないようです。

●IOポート(A port5本，B Port8本)

各ピンごとに入力あるいは出力に設定可能なピンです。出力に設定されたPort Bピンに対して，内部Pull-upを行うこともできます(ピンごとに設定ではなく，8本すべてにかけるかかけないか，です)。また，Port Bの4～7ピンの信号レベルに変化があった場合，自動的に割り込みを起こすことも可能です。

●割り込み入力ピン

このピンを信号を変化させると割り込みが発生します(エッジ検出)。極性はソフトウェアで設定できます。

■各レジスタの解説

レジスタといっても，アキュムレータのWレジスタ以外，みなRAM空間からアクセスするようになっています。メモリマップをデータシートから， PIC16F84に便利なように再構成し引用します。

表中にあるように，0x01，0x05，0x06，0x08，0x09番地はバンク切り替えにより，内容が変わりますので，アクセスする際は注意してください。

また，0x03番地のstatusレジスタとは，いわゆるフラグレジスタですが，普通のCPUとは少し動作が異なるので注意が必要です。

加算命令(addwf / addlw)では，

C=1：桁あふれが発生

C=0：桁あふれなし

ですが，減算命令(subwf / sublw)では，

C=1：桁あふれなし(演算結果が0を含む)

C=0：桁あふれが発生

と逆になっています。

■PICマイコンのアーキテクチャ

読者の皆さんがZ80か68000などのCPUに詳しいものとして，以下ではPICマイコンがそれらのCPUとどう違うのかを説明します。

●レジスタはwレジスタのみ

通常のCPUでは，加減算などの操作をレジスタ対レジスタで行いますが，PICマイコンではレジスタがひとつしかないので，wレジスタとRAM内のデータで行います。

あるアドレスValue1の値に0x12を足すという場合，Z-80では，

```
LD      HL, Valu1
LD      A, (HL)
LD      B, 0x12
ADD     A, B
LD      (HL), A
```

と，必要な値をすべてレジスタに置いてからでないと演算(この場合はADD A,B)できませんでした。これに対し，PICマイコンでは，

```
movlw  0x12
addwf  Value1, 1
```

と，メモリの内容に対して，直接，演算を行えます(1)。

別な見方をすると，RAMの内容すべてがレジスタと同様に参照できるので，Wというひとつのアキュムレータと68個の通常レジスタがあるといえなくもありません。このあたりから，Microchip Technologyのデータシートでは，RAM空間のことをレジスタファイルと呼んでいるのでしょう。

(1)わかりづらいニーモニックですが，3章の私家版アセンブラではちゃんと，

```
w = 0x12
Value1 += w
```

と記述できるようになっています。

●フラグ類もRAMにマッピングされている

0x00～4fのRAM領域のうち，0x00～0x0cは，すでにハードウェアで用途が決まっています(表1)。PIOやタイマのペリフェラルを，RAM領域から制御するというのは，68000ファミリで行われていたごく一般的なことですが，PICマイコンでは，CPUのフラグやプログラムカウンタまで，RAM領域からアクセスできるようになっています。

フラグのようにCPUにとってとても重要な情報が，ほかのRAM内容と同様に扱えてしまうというのは，ちょっと気味が悪いですが，RAMとフラグなどを統一して扱えるようにすることで，フラグ専用の命令(キャリフラグをクリアするとか，フラグ情報を一式，保存するとか)が不要になります。このような工夫の積み重ねが，PICマイコンのシンプルなアーキテクチャを可能にしたのでしょう。

●間接アドレッシング用のアドレスレジスタ

たとえば，Z80でRAMのアドレス0x12の値を参照する場合，

```
LD    HL, 0x12
LD    A, (HL)
```

と，HLレジスタをアドレスレジスタとして，間接アドレッシングを用いて値を参照します。ところが，PICマイコンにはレジスタはアキュムレータのwレジスタしかありませんので，このような使い方はできません。代わりに，

RAMのアドレス0は実在しない。代わりにFSRの値のアドレスをアクセスする

Bank 0

アドレス	名前	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00	INDF	ここにアクセスしようとすると，実際にはFSRで指定されたアドレスにアクセスする。							
0x01	TMR0	タイマのダウンカウンタ							
0x02	PCL	プログラムカウンタの下位8ビット							
0x03	STATUS	未使用		RP0	/T0	/PD	Z	DC	C
	RP0	RAMバンクの切り替え 0: Bank 0, 1: Bank1							
	/T0	直前のリセット要因 /T0 /PD							
	/PD	0 0 スリープ状態からWDTによりリセット 0 1 WDTによりリセット 不変 0 スリープ状態から外部割り込みにより起動 1 1 電源投入後							
	Z	1：演算結果が0だった 0：0でなかった							
	DC	1：加算の途中で4ビット目から5ビット目へ繰り上がりが発生した 0：発生しなかった							
	C	1：演算で桁あふれが起こった 0：起こらなかった							
0x04	FSR	アドレスレジスタ。レジスタ間接アドレッシングに使用							
0x05	PORTA	未使用			PORTAへのアクセス				
0x06	PORTB	PORT Bへのアクセス							
0x07		未使用							
0x08	EEDATA	EEPROMに書くデータ・読んだデータ							
0x09	EEADR	アクセスするEEPROMのアドレス							
0x0a	PCLATH	未使用			プログラムカウンタの上位5ビット				
0x0b	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
	GIE	1：マスクされていないすべての割り込みを許可する 0：すべての割り込みを禁止する							
	EEIE	1：EEROM書き込み終了割り込みを許可する 0：EEROM書き込み終了割り込みを禁止する							
	T0IE	1：タイマ割り込みを許可する 0：タイマ割り込みを禁止する							
	INTE	1：INTピン割り込みを許可する 0：INTピン割り込みを禁止する							
	RBIE	1：RB port変化割り込みを許可する 0：RB port変化割り込みを禁止する							
	T0IF	タイマ割り込みが発生したか？1：発生した 0：発生していない							
	INTF	INT pin割り込みが発生したか？1：発生した 0：発生していない							
	RBIF	RB port変化割り込みが発生したか？1：発生した 0：発生していない							
0x0c～0x80		ユーザーがワークエリアとして自由に使用可能							

表1：PIC16F84のRAM空間メモリマップ

というルールがあるので，

```
    movlw          0x12
    mofwf          FSR, 1
    movf 0, 0      ;0x00でなく，0x12を読む
```

と記述します[2]。

(2)再びわかりづらいニーモニックですが，

```
    w = 0x12
    FSR = w
    w = indirect
```

と私家版アセンブラでは記述できます。

●データとプログラム空間が分離

宣伝文句の「ハーバードアーキテクチャ採用，高速RISCプロセッサ」というのを見ると，プログラムコードのフェッチとRAMデータのフェッチを並列して行う最近のRISCプロセッサのように聞こえますが，実際はそうでもありません。しっかり，1命令サイクル＝4クロック(フェッチ，デコード，実行，ライトバック)がかかっていて，これらが並列に行われることはありません。

ただし，ワークエリアが0x00からの連続した下位アドレスに寄せ集められるため，7ビット幅のオペコードでも，ひととおりアクセスすることが可能になっています。

●プログラム中にテーブルが使えない

PICマイコンでは，プログラム領域とデータ領域が完全に分離されています。よって，プログラム領域のあるアドレスの値を参照する命令などありません。電源投入時にEEPROMからRAMへデータが転送するなどの対応策が必要です。

あるいは，少々トリッキーですが，List 1のようにretlw命令を用いて，近い処理をすることも可能です。

このプログラムはPCもほかの変数と同様に加算を行えるという特徴を利用したものです。ひとつ注意しなければならないのは，演算の精度が8ビットで，PCが13ビットとビット幅が異なることです。PCLになにか値を書き込むと，足りない5ビット分はPCLATHから自動的に補充されます。ですから，このようにPCLに値を書き込むときには，常にPCLATHの値を気にする必要があります。さもないと0x0123番地に飛ばしたかったのに，PCLATH＝2だと0x0223番地に飛んでしまうという事態になります。

●スタックが8段

一般のCPUでは，サブルーチンコールの戻り番地の保存にRAMを用いますが，PICマイコンでは特別なスタック用のメモリ空間があり，そこに保存されます。そのメモリ空間の大きさはファミリによって違いますが，PIC16F84の場合，13ビット×8Wordですので，8段までのサブルーチンコールを行うことができます。

表3がPIC16xxxファミリに使われている命令一覧です。命令数が全部で35と少なく，とてもシンプルにまとまっています。

- b：ビットの指定(0～7)
- f ：RAMのアドレス。(0～0x7f)
- d：演算結果をどこに代入するか
- 0：Wレジスタ
- 1：RAM
- k：8ビットの定数
- a：11ビットのアドレス

Bank 1									
0x01	Option	/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
	/RBPU	1：出力に設定されたPort B pinにプルアップ処理を行わない 0：出力に設定されたPort B pinにプルアップ処理を行う							
	INTEDG	1：INT pinの極性を立ち上がりに設定 0：INT pinの極性を立ち下がりに設定							
	T0CST0SE	内部タイマのクロック源指定　T0CS T0SE 0　x　内部クロック 1　0　T0CK pin立ち下がり 1　1　T0CK pin立ち上がり							
	PSA	プリスケーラの割り当て 1：WDT 0：タイマ							
	PS[2:0]	プリスケーラのレートを設定 タイマ　WDT 000 1/2　1/1 001 1/4　1/2 010 1/8　1/4 011 1/16　1/8 100 1/32　1/16 101 1/64　1/32 110 1/128　1/64 111 1/256　1/128							
0x05	TRISA	未使用			port Aの向き0:入力、1:出力				
0x06	TRISB	port Bの向き。 0:入力， 1:出力							
0x08	EECON1	未使用			EEIF	WRERR	WREN	WR	RD
	EEIF	EEROM書き込み終了割り込みが発生したか 0:発生してない 1: 発生した							
	WRERR	EEPROM書き込み中にエラーが発生したか 0:発生してない 1: 発生した							
	WREN	1：EEPROM書き込みを許可する 0：EEPROM書き込みを禁止する							
	WR	このビットに1を書き込むことによりEEPROMのEEADRアドレスにEEDATAの値を書き込み始める。 このビットに0を書き込んではいけない。							
	RD	このビットに1を書き込むことにより EEPROMのEEADRアドレスの値をEEDATAに読み出す。 このビットに0を書き込んではいけない。							
0x09	EECON2	EEPROM書き込みのための専用レジスタ。詳細はデータシートを参照							

== 代入命令 ==

clrf f(Clear f)

RAMのアドレスfの内容をクリア(0を代入)します。

例)

前) flag_reg = 0x5a
　clrf　　flag_reg
後) flag_reg = 0x0, Z flag = 1

clrw(Clear W)

wレジスタの値をクリア(0を代入)します。

例)

前) w = 0x5a
　clrw
後) w = 0, Z flag = 1

clrwdt(Clear WDT)

ウオッチドッグタイマのカウンタをクリア(0を代入)します。リセット後に、リセット要因を調べられるよう以下のようにフラグを設定します。

/T0←1
/PD←1

movf f, d(Move f)

d=1: RAMのアドレスfの内容を参照し、wレジスタに代入します。

d=0: RAMのアドレスfの内容を参照し、RAMのアドレスfに代入します(RAMの内容は変わりませんが、zフラグの内容が変化します)。

例)

　movf fsr, 0
後) w = *fsr

movlw k(Move literal to W)

8ビットの定数kをwレジスタに代入します。

例) movlw　　0x5a
後) w = 0x5a

movwf f(Move w to f)

wレジスタの値をRAMのアドレスfに代入します。

例) movlw　　0x12
　movwf　　0x34
　;アドレス0x34に0x12を代入

== 演算命令 ==

以下の命令は、さまざまな演算を行います。基本的に2項演算でwレジスタとRAMのアドレスfの内容との演算を行い、結果はdの値により、

d＝1:wレジスタに代入
d＝0:RAMのアドレスfに代入

します。

addlw k(Add literal and W)

wレジスタの内容と8ビット定数kの値を加算します。

例)

前) w = 0x15
　addlw　　0x10
後) w = 0x25

addwf f, d(Add W and f)

RAMのアドレスfとwレジスタの値を加算します。

例)

前)w =0x17, *fsr = 0xc2
　addwf　　fsr, 0
後)w = 0xd9, *fsr = 0xc2(変化なし)

andlw k(AND literal and W)

wレジスタの内容と8ビット定数kの論理積を求めます。

例)

前) w = 0xa3
　andlw　　0x5f
後) w=03

andwf f, d(AND W and f)

RAMのアドレスfとwレジスタの値の論理積を求めます。

例)

前) w = 0x17, fsr = 0xc2
　andwf　　fsr, 1
後) w=0x17(変化なし), fsr=0x02

comf f, d(Complement f)

RAMのアドレスfの値の2の補数を求めます。

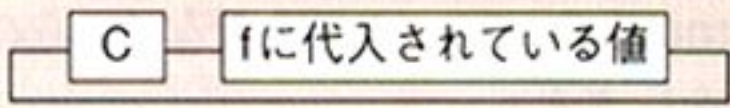
例)

前) *reg1 = 0x13
　comf　reg1, 0
後) *reg1 = 0x13(変化なし), w = 0xec

iorlw k(inclusive OR literal with W)

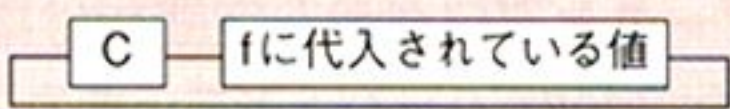
wレジスタの内容と8ビット定数kの値の論理和を求めます。

例)

前) w = 0x9a
　iorlw　　0x35
後) w = 0xbf

iorwf f, d(inclusive OR W with f)

RAMのアドレスfとwレジスタの値の論理和を求めます。

例)

前) *result = 0x13, w = 0x91
　iorwf　　result, 0
後) *result = 0x13, w = 0x93

rlf f, (Rotate Left f through Carry)

RAMのアドレスfの値を以下のように左シフトします。 [C | fに代入されている値]

例)

前) *reg1 = 0xe6, C Flag = 0
　rlf　　reg1, 0
後) *reg1 = 0xe6, C Flag = 1, w = 0xcc

rrf f, d(Rotate Right f through Carry)

RAMのアドレスfの値を以下のように右シフトします。 [C | fに代入されている値]

例)

前) *reg1 = 0xe6, C Flag = 0
　rlf　　reg1, 0
後) *reg1 = 0xe6, C Flag = 0, w = 0x73

sublw k(Subtract W from literal)

8ビット定数kの値からwレジスタの内容を減算します。

例1)

前) w = 1
　sublw　　0x02
後) w = 1, C Flag = 1, Z Flag = 0

例2)

前) w = 2
　sublw　　0x02
後)w = 0, C Flag = 1, Z Flag = 1

例3)

前) w = 3
　sublw　　0x02
後)w = 0xff, C Flag = 0, Z Flag = 0

subwf f, d(subtract W from f)

RAMのアドレスfからwレジスタの値を減算します。

例1)

前) *reg1=3, w=2
　subwf　reg1, 1
後) *reg1=1, w=2, C Flag=1, Z Flag= 0

例2)

前) *reg1=2, w=2
　subwf　reg1, 1
後) *reg1=0, w=2, C Flag=1, Z Flag=1

例3)

前) *reg1=1, w=2
　subwf　reg1, 1
後) *reg1=0xff, w=2, C Flag=0, Z Flag=0

swapf f, d(Swap nibbles in f)

RAMのアドレスfの内容を参照し、上位4ビット、下位4ビットを入れ替えます。

例)

前) *reg = 0xa5
　swapf　　reg, 0
後) *reg = 0xa5, w = 0x5a

xorlw k(exclusive OR literal with W)

wレジスタの内容と8ビット定数kの値の排他的論理和を求めます。

例)

```
前)w = 0xb5
        xorlw      0xaf
後)w = 0x1a
```

xorwf f, d(exclusive OR W with f)

RAMのアドレスfとwレジスタの値の排他的論理和を求めます。

例)

```
前)＊reg = 0xaf, w = 0xb5
        xorwf      reg, 1
後)＊reg = 0x1a, w = 0xb5
```

decf f, d (dec f)

RAMのアドレスfから1を引きます。

例)

```
前)cnt = 0x01
        decf       cnt, 1
後)cnt = 0, Z Flag = 1
```

incf f, d (inc f)

RAMのアドレスfに1を足します。

例)

```
前)     cnt = 0xff
        ncf        cnt, 1
後) cnt = 0, Z Flag = 1
```

== ビット演算 ==

bcf f, b(Bit Clear f)

RAMのアドレスfの、bビット目をクリアします(0にする)。

例)

```
前)flag_reg = 0xc7
        bcf        flag_reg, 7
後)flag_reg = 0x47
```

bsf f, b(Bit Set f)

RAMのアドレスfの、bビット目をセットします(1にする)。

例)

```
前)flag_reg = 0x0a
        bsf        flag_reg, 7
後)flag_reg = 0x8a
```

== 条件付き命令 ==

decfsz f,d(Decrement f, Skip next instruction if result is 0)

RAMのアドレスfから1を引きます。引いた結果が0なら、次に続く命令を無視します。

例)

```
前)cnt = 0x5a
Loop:   nop
        decfsz     cnt, 1
    goto Loop
```

後)ループを0x5a回繰り返します。

incfsz f, d(Increment f, Skip next instruction if result is 0)

RAMのアドレスfに1を足します。足した結果が(255をオーバフローして)0なら、次に続く命令を無視します。

btfsc f, b(Bit Test, Skip if Clear)

RAMのアドレスfのビットbが0ならば、次に続く命令を無視します。

例1)

```
  前)   flag [1] = 0
        btfsc      flag, 1
flag1:  goto       flag1
flag0:  goto       flag0
  後)   pc = flag0
```

例2)

```
  前)   flag [1] = 1
        btfsc      flag, 1
flag1:  goto       flag1
flag0:  goto       flag0
  後)   pc = flag1
```

btfss f, b(Bit Test, Skip if Set)

RAMのアドレスfのビットbが1ならば、次に続く命令を無視します。

例1)

```
  前)   flag[1] = 0
        btfsc      flag, 1
flag0:  goto       flag0
flag1:  goto       flag1
  後)   pc = flag0
```

例2)

```
  前)   flag[1] = 1
        btfsc      flag, 1
flag0:  goto       flag0
flag1:  goto       flag1
  後)   pc = flag1
```

== その他 ==

call a(Call Subroutine)

サブルーチンをコールします。サブルーチンのアドレスは,

(pclach [4:3]<< 11) + a (11ビット)

で決まります。

goto a(Go to Address)

以下の式により求められる値をPCに代入します。

(pclach [4:3]<< 11)+ a (11ビット)

retfie(Return from Interrupt)

割り込み処理ルーチンからリターンします。

retlw k(Return with literal in W)

kをwレジスタに代入し、サブルーチンからリターンします。

return(Return from Subroutine)

サブルーチンからリターンします。

sleep(Go into standby mode)

スタンバイモードに入ります。主だったクロックを止めるので、消費電力を下げることができます。割り込みあるいはリセットが入り次第、通常の動作モードに戻ります。

リセット後、リセット要因を調べられるように、以下のようにフラグが設定されます。

/T0←1
/PD←0

nop(No Operation)

なにもしません。ただ、1Word分メモリを消費し、1サイクル実行時間を消費します。PICマイコンの処理を一定時間停止させたい場合に使います。

== PIC16C5xの互換用命令 ==

以下の命令は、PIC16C5xとの互換性を保つために実装されています。PIC16F84では、RAMのバンク切り替えで、TRISレジスタ、OPTIONレジスタともに、通常のRAM領域としてアクセスできるようになったので、以下の命令はなるべく使わないでください。将来の新デバイスではサポートされない可能性があります。

TRIS f(Load TRIS Register)

wレジスタの値をTRISレジスタに代入します。fの値により,

f =5: TRISA
f =6: TRISB
f =7: TRISC (PIC16F84では未実装)

へ代入します。それ以外の値を設定してはいけません。

OPTION (Load OPTION Register)

wレジスタの値をOPTIONレジスタに代入します。

参考文献

(1) “PIC16F8X 18-pin Flash/EEPROM 8-bit Microcontrollers” Data sheet, Microchip Technology
(2) 野澤 康夫「パラレルポート接続の新PICライタの製作」, トランジスタ技術1997年8月号 p.311-326, CQ出版

ニーモニック	説明	実行サイクル	14-bitオペコード	変化するフラグ	注
バイト単位の操作					
ADDWF f,d	Wとfの加算	1	00 0111 dfff ffff	C, DC, Z	1,2
ANDWF f,d	Wとfの論理積	1	00 0101 dfff ffff	Z	1,2
CLRF f	fをクリア	1	00 0001 1fff ffff	Z	2
CLRW	Wをクリア	1	00 0001 0xxx xxxx	Z	
COMF f,d	fの補数	1	00 1001 dfff ffff	Z	1,2
DECF f,d	fのデクリメント	1	00 0011 dfff ffff	Z	1,2
DECFSZ f,d	fをデクリメントし,結果が0なら,次の命令を飛ばす	1(2)	00 1011 dfff ffff		1,2,3
INCF f,d	fのインクリメント	1	00 1010 dfff ffff	Z	1,2
INCFSZ f,d	fをインクリメントし,結果が0なら,次の命令を飛ばす	1(2)	00 1111 dfff ffff		1,2,3
IORWF f,d	Wとfの論理和	1	00 0100 dfff ffff	Z	1,2
MOVF f,d	fの値を代入	1	00 0000 1fff ffff	Z	1,2
MOVWF f	Wをfに代入	1	00 0000 0xx0 0000		
NOP	なにもしない	1	00 0000 0xx0 0000		
RLF f,d	左シフト	1	00 1101 dfff ffff	C	1,2
RRF f,d	右シフト	1	00 1100 dfff ffff	C	1,2
SUBWF f,d	fからWを減算	1	00 0010 dfff ffff	C, DC, Z	1,2
SWAPF f,d	fの上位4ビットと下位4ビットを交換	1	00 1110 dfff ffff		1,2
XORWF f,d	Wとfの排他的論理和	1	00 0110 dfff ffff	Z	1,2
ビット操作命令					
BCF f,b	fのビットbをクリア	1	01 00bb bfff ffff		1,2
BSF f,b	fのビットbをセット	1	01 01bb bfff ffff		1,2
BTFSC f,b	fのビットbが0なら,次の命令を飛ばす	1(2)	01 10bb bfff ffff		3
BTFSS f,b	fのビットbが1なら,次の命令を飛ばす	1(2)	01 11bb bfff ffff		3
定数演算と分岐命令					
ADDLW k	Wとkの加算	1	11 111x kkkk kkkk	C, DC, Z	
ANDLW k	Wとkの論理積	1	11 1001 kkkk kkkk	Z	
CALL k	アドレスkをサブルーチンをコール	2	10 0kkk kkkk kkkk		4
CLRWDT	WDTをクリア	1	00 0000 0110 0101	/T0, /PD	
GOTO k	アドレスkに飛ぶ	2	10 1kkk kkkk kkkk		4
IORLW k	Wとkの論理和	1	11 1000 kkkk kkkk	Z	
MOVLW k	Wにkを代入	1	11 00xx kkkk kkkk		
RETFIE	割り込み処理ルーチンからリターン	2	00 0000 0000 1001		
RETLW k	Wにkを代入し,サブルーチンからリターン	2	11 01xx kkkk kkkk		
RETURN	サブルーチンからリターン	2	00 0000 0000 1000		
SLEEP	スリープモードに入る	1	00 0000 0110 0011	/TO, /PD	
SUBLW k	kからWを減算	1	11 110x kkkk kkkk	C, DC, Z	
XORLW k	Wとkの排他的論理和	1	11 1010 kkkk kkkk	Z	

表3 PIC16F84で使える命令一覧

注：
1: I/Oレジスタがアクセスされた場合（例：MOVF PORTB, 1），I/Oピンの状態が値として使用される。たとえば，入力モードに設定されたピンが0[V]になっていた場合，このピンに対応するビットを読み出すと，0となる。
2: プリスケーラがタイマに使用されている場合，これらの命令を用いてTMR0レジスタに書き込みを行うと，プリスケーラの値はクリアされる。
3: 条件が成立した場合，次の命令を無視するのに1サイクルかかる。
4: 飛び先のアドレスは，
上位5ビット＝PCLATHレジスタ
下位8ビット＝命令の引数
から構成される。

Bit [13:5]	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
未使用	CP	/PWRITE	WDTE	F0SC1	F0SC0
CP	プログラムにプロテクトをかけるか？ 1：かける 0：かけない				
/PWRITE	パワーアップタイマを使用するか？ 1：使用しない 0：使用する				
WDTE	ウオッチドッグタイマを使用するか？ 1：使用する 0：使用しない				
FCSC1	使用する外部クロックの種類 F0SC1 F0SC0 1 1 RC RC発信器				
F0SC0	1 0 HS 4 - 10MHzの水晶 0 1 XT 100 - 4MHzの水晶 0 0 LP 32 - 200 kHzの水晶				

表2 PIC16F84のコンフィグレーションワード（プログラム空間の0x2007）

ウオッチドッグタイマとは

プログラムのバグには，たちの悪いものがあって，システムを連続10時間動かさないと発生しないものがあります。また，プログラムのバグでなくても，ある条件下で電源が瞬断したり，外部ノイズによりCPUがあらぬ動作を始めてしまう場合があります。

このようなバグの被害を最小限にとどめる方法のひとつにウオッチドッグ（番犬）タイマがあります。原理は簡単で，システムに一定時間でリセットをかけるというものです。たいていのシステムでは，リセット後の初期化は，システムがどんな状態になっていても，既知の状態に設定するようになっていますから，なにかの理由でCPUが暴走しても，リセットにより，最初からやり直すこともできますし，常にリセット後数分内の状態に保つことができますので，数十時間に1回しか発生しないバグも，極力抑えられるようになります。

バグがないプログラムを書くのが理想ですが，なにごとにも「絶対」という保証はありません。二重，三重に安全策を講じたいとき，ウオッチドッグタイマが役に立ちます。

List 1 ： 0x00～0x0fの値をASCIIキャラクタに変換するプログラム

```
pcl         equ         2
Bin2Hex:
            addwf       pcl,1; pc = pc+1 + w
            retlw       '0'
            retlw       '1'
            retlw       '2'
            retlw       '3'
            retlw       '4'
            retlw       '5'
            retlw       '6'
            retlw       '7'
            retlw       '8'
            retlw       '9'
            retlw       'A'
            retlw       'B'
            retlw       'C'
            retlw       'D'
            retlw       'E'
            retlw       'F'
```

第4章 PIC16xxx用オリジナルアセンブラPASM

高尾克彦 Takao Katsuhiko

PICの概要と命令，そしてアセンブラプログラムを紹介した。あとは実践あるのみだ。ここではPICマイコンを使用したタイマ付きAC電源制御回路を作ってみよう。エアチェックタイマ以外にも機器制御などでいろいろ応用ができるはずだ。

■ニーモニックを拡張する

第3章で，ひととおり純正のニーモニックを見てきたわけですが，これで，すぐにプログラミングを始めることのできる方は少ないのではないかと思います。確かに35語と覚えるべき命令数は少ないですが，その1つひとつが変な英語の略語になっていて，これらを組み合わせてプログラムを作るのは大変です。

ある程度複雑なアルゴリズムを実装するには，ニーモニックを見ただけで，その動作がわからなければなりませんが，PICマイコンの場合，そのレベルまで達するにはとても時間がかかります。

これはPICマイコンのアーキテクチャが複雑で扱いづらいのではなく，単にニーモニックに問題があると思います。私が使いやすいようなニーモニックを扱えるアセンブラを作りましたので，ここに発表させていただきます。

とはいっても，すべてを定義し直すのではなく，純正のニーモニックと共存する形で拡張します。PICマイコンがほかの組み込み用マイコンと大きく違うのは，大口ユーザーだけでなく，われわれアマチュアユーザーでも同社のホームページを通じてサポートが受けられるという点です。Microchip Technologyは，安価な開発ツールを用意するだけでなく，ホームページからダウンロードできるさまざまなアプリケーションノートとサンプルプログラムを用意しています。また，最近では，各地にPICマイコンユーザーズクラブや各種メーリングリストなどが組織されていて，それらのホームページにもさまざまなプログラムが掲載されています。

多くの場合，これらのプログラムは純正ニーモニックで書かれていますので，新しいアセンブラもこれらを利用できるようにします。さもないと，せっかくのプログラムなのに，すべての命令を書き直さなければなりません。これは非常に手間のかかることですし，不要なバグの元です。ですから，他人のプログラムから引用する部分は，手を加えなくてよいよう純正のニーモニックを使い，自分で書く部分はオリジナルのニーモニックで書くということができます。

■演算結果の代入先

純正ニーモニックをわかりにくくしているいちばんの原因は，＊＊＊fw命令の，<d>フィールドでしょう。0で，演算結果をwレジスタに代入し，1でRAMに代入します(第3章参照)。まず，これが頭に入っていないとプログラムを書くどころか，人のプログラムも読めません。

なぜわかりづらいかというと，このルールになんの必然性もなければ，実世界との対応もないからです。

たとえば，Z80では，

```
add        a,h
```

はhレジスタの値をaレジスタの値と加算し，結果をaレジスタに代入するニーモニックでした。Z80では，演算結果は，第1引数(左側)に代入されるというルールで統一されていましたので，覚えるのはそれほど苦でもありませんでした。

ところが，この右から左へというルールにすべての人が同意するかといえば，難しいところで，68000では，

```
add       d0,d2
```

では，d0レジスタの内容をd2レジスタの内容を加算し，結果をd2レジスタに代入するというように左から右へと動かします。

ひとつのプロセッサに対し，ひとつのルールを決めてしまえば少しは楽になりますが，右から左のルールをPICマイコンに適用すれば，昨日まで68Kのプログラムを作っていた人には不自然でしょうし，逆にするとZ80のプログラムを作っていた人は不自然に感じるに違いありません。

そこで，PASMは，我々がもっとも親しんでいる算術記法を用いることにしました。つまり，

```
代入先＝引数1<演算子>引数2
```

という書式です。

いままでアセンブラに親しんでこられた方には，ちょっと違和感があるかもしれませんが，純正のニーモニックよりはわかりやすくなったと思います。

```
w＝w + value
```

PICマイコンにwレジスタがあることを知らなくても，この意味がわからない人はいないでしょう。

■オペコードを統一する(clr系)

次に，初心者を悩ませるのが，オペコードの種類です。言語処理系を作る立場からいうと，アドレッシングモードが少なければ，オペコードはいくらあっても，負担にならないのですが，こんなところで楽をしてはいけません。

まず，いちばん目につきやすいclr??系から。

```
clrf  f      fの値をクリア
clrw         wレジスタをクリア
clrwdt       WDTをクリア
```

これは，引数の種類だけ，オペコードを拡張してしまった悪い例です。幸い，〝clr〟という文字列は純正ニーモニックに使われずに無傷でいるので，さっそく，〝clr〟で統一しましょう。上から，

```
clr        f
clr        w
clr        wdt
```

これでユーザーが覚えるべき命令が2つ減りました。clrとは，値を0にすることですから，ついでに，

```
f＝0
```

という書式も扱えるようにします。ただし，w＝0というのは，あとで出てくるmovlw 0に取っておくため，ここではあえて扱えないようにしておきます(clrwとmovlw 0ではzフラグの変化が違うので，完全に統一することはやめました)。

■オペコードを統一する(代入系)

先ほど，演算結果をどこに代入するかを「＝」で指定できるようにしましたが，データの代入も同じように扱えるようにしましょう。

```
movf f,d
  fの値をwレジスタに代入(d＝0のとき)
  fの値をfに代入(d=1の時，フラグが値によって変化する)
movwf
```

wレジスタの値をfに代入
　movlw　　k
　wレジスタにkを代入

これも引数の数だけ，命令がありますので，以下のように「=」に統一しましょう。

　w = f
　f = f
　f = w
　w = k

これで，mov??系の命令を使う必要がなくなりましたので，ユーザーが覚えるべき命令が3つ減りました。

■オペコードを統一する（演算系）

PICマイコンには加算・減算・論理和・論理積・排他的論理和などの計算機能がありますが，これが定数を用いて行うか，RAM上の値を用いるかによって，以下のようにオペコードが変わります。

	RAM	定数
加算	addwf f,d	addlw　k
減算	subwf f,d	sublw　k
論理積	andwf f,d	andlw　k
論理和	iorwf f,d	iorlw　k
排他的論理和	xorwf f,d	xorlw　k

「add」とか「sub」などはよいとして，その後ろに「wf」「lw」をつけて，引数の種類を指定しなければいけないのは，二度手間です。本来なら，アセンブラが面倒を見るべきところです。と，そのようなインプリメンテーションにしてもよいのですが，せっかく代入記号「=」を導入したので，さらにわかりやすく，

	RAM（d=0の場合）	定数
加算	f = f + w	w = k + w
減算	f = f - w	w = k - w
論理積	f = f & w	w = k & w
論理和	f = f ¦ w	w = k ¦ w
排他的論理和	f = f ^ w	w = k ^ w

	RAM（d=1の場合）
加算	f = f + w
減算	f = f - w
論理積	f = f & w
論理和	f = f ¦ w
排他的論理和	f = f ^ w

と表記できるようにします。減算以外は引数1と引数2を入れ替えても結果は同じですので，

	RAM（d=0の場合）	定数
加算	f = w + f	w = w + k
論理積	f = w & f	w = w & k
論理和	f = w ¦ f	w = w ¦ k
排他的論理和	f = w ^ f	w = w ^ k

	RAM（d=1の場合）
加算	f = w + f
論理積	f = w & f
論理和	f = w ¦ f
排他的論理和	f = w ^ f

という表記も一緒に実装します。

ここで，「f = w - f」という命令がないのは，PICマイコン自身にそのような機能がないからです。私も驚きましたが，本当に，減算は「f = f - w」しかないようです。このようにした理由は私には全然わかりませんが，どなたか，ご存じの方いらっしゃいますか？

「subwf f,d」や「sublw k」などと書いていては，この点をうっかり見落としがちですが，「f = f - w」「f = k - w」と書くことで，注意が届くようになります。

それはともかく，これでユーザーが覚えるべき命令が5つ減りました。

■条件分岐

これで，35あった命令を3割近く減らしました。しかも，演算子をC言語と統一したので，かなり垣根が低くなったと思います。この20個弱の命令でもプログラムは作れるのですが，今度は，いくつかの重要な命令が見あたりません。

意外に思われるかもしれませんが，PICマイコンには条件分岐命令がありません。条件分岐命令とは，たとえば「キャリフラグが1なら，アドレスLabelへ飛ぶ」という命令で，Z80なら，

　JP　　　C, Label

で，68000なら，

　BCS　　Label

という命令です。

その代わり，PICマイコンには，アドレスfのビットbが0（あるいは1）ならば次の命令を無視するという

　BTFSC f,b　Bit Test f, Skip it Clear
　BTFSS f,b　Bit Test f, Skip it Set

という2命令があります（英文は参考文献1より，そのまま引用）。

さらにPICマイコンは，フラグ情報やPCがRAM上から見えるという特徴がありますから，「キャリフラグが1なら，アドレスLabelへ飛ぶ」という命令は「RAMのアドレス3のビット0（キャリフラグ）が0なら，アドレスLabelへ飛ぶという

Boxed Article

インテルニーモニックとザイログニーモニック

その昔，インテル社の8080というデバイスがありまして，一応，このデバイスを持って，現在のマイコン産業が始まったということになっているようです。このCPUは，いま考えると，＋5Vと＋12Vの2電源必要だわ，φ1とφ2の2相クロックが必要だわ，肝心な制御信号がデータバスとマルチプレクスされているわ，というそれはそれは使い辛いCPUでした。昔の人は我慢強かったなぁ。

このCPUのニーモニックは，時代が時代ですから，本文中でいっていたように，引数の種類だけ異なる命令があるというニーモニック体系を用いていました。たとえば，HLレジスタのアドレスからAレジスタに8ビットの値を代入するのが，

　MOV　　　A,M

という命令です（HレジスタとLレジスタをまとめて，Mレジスタと呼ぶ）。で，HLレジスタの代わりにBCレジスタを使おうとすると，

　LDAX B

となり，以下，面倒なので，Z80のニーモニックで説明すると，

```
mvi a,0x12          ld a, 0x12
lxi b,0x1234        ld bc,0x1234
mov m,a             ld (hl),a
mvi m,0x12          ld (hl),0x12
stax b              ld(bc),a
lhld 0x1234         ld hl,(0x1234)
shld 0x1234         ld (0x1234),hl
```

と，Z80のLD命令だけでも，7種類もプログラマが使い分けなければなりませんでした。ま，時代が時代ですから，アセンブラが1分間に何行処理できるとか，メモリが4Kバイトしかないシステムで動かなければならない，とかいろいろな制約の下にこうなったようです。

そういえば，4MHzのZ80で2000行近いプログラムをアセンブルすると，だいたい10分くらいかかっていました。きっと，インテルニーモニックでかけば3分くらいで済んだのでしょうね。

次の命令を無視する」と実現します。いい換えると「キャリフラグが1なら，アドレスLabelへ飛ぶという次の命令を実行する。それ以外は無視する」ということになります。具体的には，

```
btfsc   3,0
goto    Label
```

と書きます。

値の大小判定は，ただでさえ考えることが多いのに，その裏命題を考えてないと，分岐命令が使えないというのは，とても使いづらいです。そこで，上の命令を，

```
if c goto Label
```

と書けるようにします。キャリフラグだけでなく，デジットキャリフラグ(ボローフラグ)，ゼロフラグもありますし，それらのフラグが0のときに分岐したいこともあるでしょうから，

```
if c goto Label
if nc goto Label
if dc goto Label
if ndc goto Label
if z goto Label
if nz goto Label
```

を実装します。

よく考えると，2命令のうち，2つ目は「goto」文でなくてもよいので，すべての命令が使えるようにしておきます。

例)if z return

```
if nc call Label
if c f = f + 1
```

●w=w+resultは，w=w+constant？ それともw = w + fr？

たとえば，

```
w = w + result
```

という命令があったとしましょう。これまで読んでこられた方なら，即座にwレジスタに，RAMのアドレス「result」の内容を足す命令だ，とすぐに気づかれることでしょう。

さらに，鋭い方は，resultというラベルはRAMのアドレスではなく，ただの定数10で，

```
w = w + 10
```

という意味かもしれない，と思うかもしれません。

両方とも正解です。

これは，アドレスfの値をwレジスタに足すという「addfw f,d」と定数kをwレジスタに足す「addlw k」という命令を同じフォーマットに統一してしまったから起こる不明瞭さです。実際の動作は，resultがどこで定義されているかにより決まり，

(1)resultがRAMのアドレスだった場合，

```
        segment   ram
        org       0x0c
result:   ds              1
        segment   program
        w = w + result
```

は，

```
addfw f,0
```

と同じ意味です。

(2)resultがRAMのアドレスでない場合

```
result      equ       10
    segment   program
    w = w + result
```

は，

```
addlw     f,10
```

と同じ意味です。

普通，アセンブラレベルでは，このような切り分けは明示的に行われますがPASMではあえて自動的に行うようにしました。過去に68000で嫌な目にあったのが理由です。

```
move.w    0x640,d1
```

は0x640をd1レジスタに代入ではなくて，アドレス0x640の値をメモリから読み込んでd1レジスタに代入，という意味です。0x640をd1レジスタに代入したいときには，

```
move.w    #0x640,d1
```

と数値の前に「#」をつけなければなりませんでした。この「#」を忘れては，「アドレスエラーが発生しました」とX68000に怒られていました。

実は，秋月電子のオリジナルアセンブラ「pa.exe」も，この，

#があれば，即値*1
#がなければ，間接アドレッシング*2

というルールを適用していました。で，やはり，プログラムを作ると，wレジスタに3を足そうと思ったら，RAMの3番地をアクセスしてしまったり，と，わけのわからないエラーに苦しめられ

Boxed Article

算術表記のアセンブラ

アセンブラに"＝"や"＋"という記号を使うということが，とても新鮮に感じる方がいるかもしれませんが，残念ながら，この表記はPASMが初めてではありません。私の知っている限りでは，Carry LabというソフトハウスのBASEというZ-80用アセンブラが，

```
HL = HL + HL
```

とか，

```
E = L
```

という表記を用いていました。ただ，算術以外の表記も，英略語ニーモニックの使用を避け，特殊記号を割り当てたため，私には使いづらく感じました。たとえば，以下はHレジスタ×Lレジスタ→HLレジスタの演算を行うサブルーチンです。

```
MUL8 [DE E=L
 L=0 D=0
 Do B=8
  HL=HL+HL
  IF CY = 1 THEN HL = HL + DE
 UNTIL DEC(B)= 0
 ]DE
 RETURN
```

ここで，

```
[DE    PUSH    DE
]DE    POP     DE
```

です。このほかにも，

```
DE<>HL         EX       DE,HL
```

などと，ちょっとやりすぎた感じがします。ザイログ純正のアセンブラフォーマットをサポートしていなかったため，どちらからか，好きなほうをひとつ選ばなければならず，結局私はずっと純正のニーモニックを使っていました。

また，最近ではアナログデバイスのADSP2100シリーズが同様のニーモニックを採用し，在野の数値演算野郎の福音となっています。

```
fir:    MR=0, MX0=DM(I0,M1)MY0=PM(I4,M5);
          DO sop UNTIL CE;
sop:      MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM
(I4,M5);
          MR=MR+MX0*MY0(RND);
if MV SAT MR;
        RTS
```

純正ニーモニック	Level 1	Level 2a		Level 2b	
		d=0	d=1	d=0	d=1
addwf f,d		w = w + fw = f + w	f = w + ff = f + w	w += f	f += w
andwf f,d		w = w & fw = f & w	f = w & ff = f & w	w &= f	f &= w
clrf f	clr f		f = 0		
clrw	clr w				
comf f,d	neg f	w = ˜f	f = ˜f		
decf f,d		w = f - 1	f = f - 1		f--
decfsz f,d					
incf f,d		w = f + 1	f = f + 1		f++
incfsz f,d					
iorwf f,d		w = w ¦ fw = f ¦ w	f = w ¦ ff = f ¦ w	w ¦= f	f ¦= w
movf f,d		w = f	f = f		
movwf f			f = w		
nop					
rlf f,d		w = f << 1	f = f << 1		f <<= 1
rrf f,d		w = f >> 1	f = f >> 1		f >>= 1
subwf f,d		w = f - w	f = f - w		f -= w
swapf f,d		w = swap(f)	f = swap(f)		
xorwf f,d		w = w ^ fw = f ^ w	f = w ^ ff = f ^ w	w ^= f	f ^= w
bcf f,b	clr f,b				
bsf, f,b	set f,b				
btfsc f,b					
btfss f,b					
addlw k		w = w + kw = k + w		w += k	
andlw k		w = w & kw = k & w		w &= k	
call k					
clrwdt	clr wdt				
goto k					
iorlw k		w = w ¦ kw = k ¦ w		w ¦= k	
movlw k		w = k			
retfie					
retlw k					
return					
sleep					
sublw k		w = k - w			
xorlw k		w = w ^ kw = k ^ w		w ^= k	
純正ニーモニック	Level 2c				
addwf 2,1	goto pc + w				
addwf 2,1	goto w + pc				
movwf 2	goto w				
xorlw 0xff	w = ˜w				
bcf 3,5	bank 0	RAMバンクの設定			
bsf 3,5	bank 1	RAMバンクの設定			
純正ニーモニック	Level 3				
decfsz fr,1goto k	djnz fr,k				
btfsc 3.2{instruction}	if z { instruction }	条件付き実行			
btfss 3.2{instruction}	if nz{ instruction }	条件付き実行			
btfsc 3.1{instruction}	if dc{ instruction }	条件付き実行			
btfss 3.1{instruction}	if ndc{ instruction }	条件付き実行			
btfsc 3.0{instruction}	if c { instruction }	条件付き実行			
btfss 3.0{instruction}	if nc{ instruction }	条件付き実行			

表1　純正ニーモニックと拡張ニーモニック

ました。

そんな経験から，この切り分けは，アセンブラが自動的に行うようにしました。とはいっても，RAMのアドレスを即値として使いたい場合もあるかもしれません。そんなときは，ラベルの前に，「&」とつけてください。これで，

w = & label

は，

movwf　label,0（RAMのアドレスlabelの内容をwレジスタに代入）

でなく，

movlw　label（wレジスタにlabelを代入）

と同じ意味になります。

(1)定数そのもの。メモリをアクセスせずにそのまま使用することから。Immediate Valueの訳語。
(2)メモリの指定されたアドレスから値を参照すること。Indirect Addressingの訳語。

＊　＊　＊

と，以上の考察を踏まえて，独自に拡張したのが表1です。これでプログラミングがかなり楽になるはずですが，命令の名前を変えただけですので，デバイスそのものの性能が上がるわけではありません。後は，あなたのプログラミングの腕次第です。

■使い方

プログラムをテキストエディタで入力し，MS-DOS Promptから，

C:> PASM filename.asm

のように入力するとアセンブリを開始します。このとき，入力されたプログラムに問題がなければ，

0 Error(s), 0 Warning(s)

と画面に表示され，

filename.hex
filename.lst

という2種類のファイルが出力されます。HEXファイルはプログラムライタに渡すアセンブリ結果で，インテルHEXフォーマットで出力されています。LSTファイルは，出力結果のみならず，入力された結果もまとめて出力されており，付属のソースコードデバッガに使用します。

アセンブル時には，以下のオプションが使用可能です。

-e

EEPROMへ渡すデータのアドレスを指定します。私のAll-07では，0x2100をEEPROMへ書き込むデータのアドレスと扱うので，これをデフォルトとしましたが，秋月のAKI-PICキットを使う方は，

PASM -e 8000 filename.asm

のように0x8000を指定してください。

-r

RAMへ渡すデータのアドレスを指定します。デフォルトの値は，0x2100。

-c

コンフィグレーションROMへ書き込むデータのアドレスを指定します。デフォルトは，Microchip Technologyのデータシートで規定されているように0x2007ですが，一部のプログラムライタでは異なるようなので，各自プログラムライタのマニュアルを見ながら指定してください。

■PSIM6の使い方

アセンブラだけあっても，つまらないので，取り急ぎシミュレータを作ってみました。ブレークポイントやレジスタの値の更新ができないという，非常に初期的な状態ですが，自分の書いたプログラムに従って，メモリやレジスタの値が変わっていくのをみると，学習が早まるのではないかと思って，自分のWindowsプログラミングテクニックも省みず，作ってみました(菊池功君，MFCについていろいろ教えてくれてありがとう)。

まず，PSIM6を起動し，File ¦ Openで，適当な*.lstファイルを選択します。そのあと，Windowメニューの下にさまざまな情報を表示するウィンドウを開くためのメニューがありますので，それを使って必要なウィンドウを開いてください。

用意ができたら，

F5でRUN
F10でStep（サブルーチン内はチェックしない）
F11でTrace（サブルーチン内もチェックする）

で，プログラムを実行してみてください。

運悪く，無限ループなどでなにも反応がない場合には，Debug ¦ Stopで，プログラムの実行を止めてください。そのとき，PICマイコンはどの位置を実行していたのかが表示されます。

また，このプログラムでは，EEPROMを除くペリフェラルや割り込みをまだサポートしていません。

■PSIM6のわかっているバグ

●すでにソースファイルを開いている場合でも，さらにCtrl + Oで，ファイルが開けてしまう。
●ソースファイルを開いていないと，各種メモリウィンドウが開けない。
●プログラムを"Run"あるいは"Trace"中，ツールバー用のバルーンヘルプが表示されない。

参考文献
1: "Microchip Technology "PIC16F8x 18- bit Flash/EEPROM 8-bit Micro controllers "1998

Boxed Article

インテルHEXフォーマット

PASMは，アセンブル結果をインテルのHEXファイルで出力します。このほかにも以前はモトローラ社の提唱するSフォーマットというのがありましたが，現在では，HEXフォーマットが組み込み機器開発システムで用いられるファイルフォーマットの標準となっているみたいです(一応)。

これは，図1のようにやや冗長な構成になっています。通常のマイコンプログラムをROMに書き込む場合は，単なるバイナリファイルでもよいのですが，PICマイコンは，プログラム本体のほかに，EEPROMデータも扱わなければなりませんし，コンフィグレーションデータも一度に書き込めたほうが便利です。これらは，とびとびのアドレスに存在しますので，ベタファイルで扱っていては，空白のアドレスもなんらかのデータを書き込むことになって，時間の無駄となってしまいます。

以上のような理由で，PASMでは，アセンブル結果をバイナリファイルではなくインテルHEXフォーマットで出力することにしました。

```
: 10 0000 00 10071007100710079007900790079007 38
```

- 1文字目　常に":"。インテルHEXフォーマットであることを示す。
- 2～3文字目　この行に何バイトのデータがあるか
- 4～7文字目　アドレス
- 8～9文字目　00:通常のデータ
- 10～文字目　実際のデータ（バイト単位）
- チェックサム

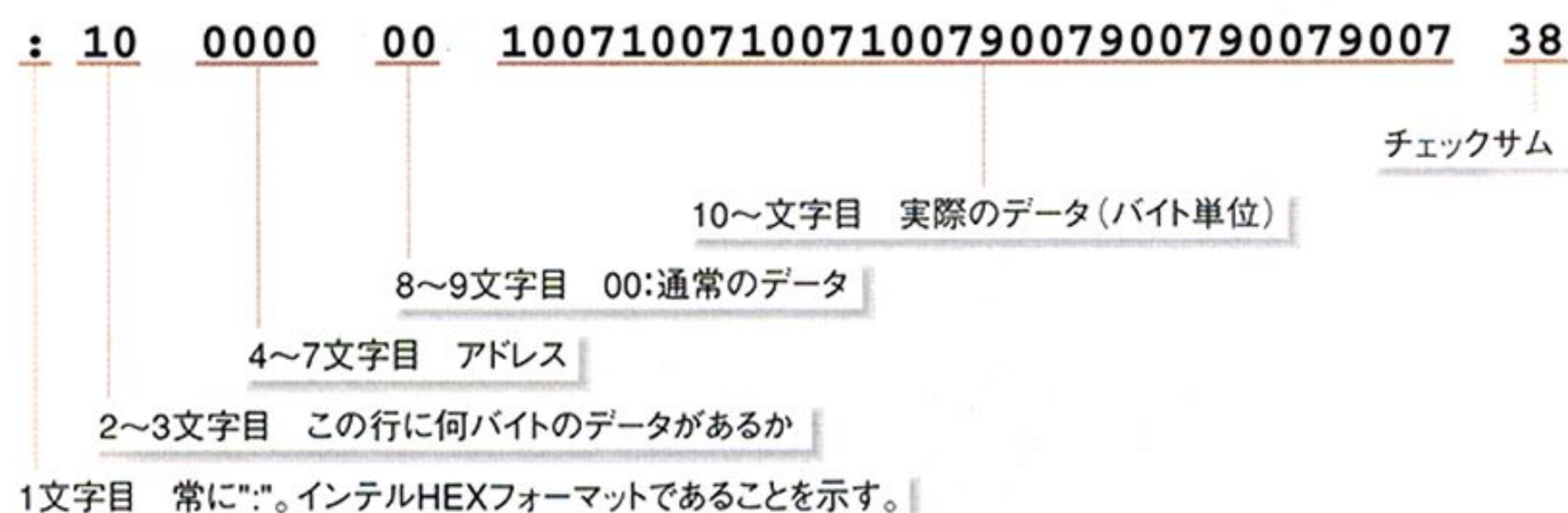



図1　インテルHEXフォーマット

第5章 オリジナルエアチェックタイマを作る

石上達也 Ishigami Tatsuya

PICの概要と命令，そしてアセンブラプログラムを紹介した。あとは実践あるのみだ。ここではPICマイコンを使用したタイマ付きAC電源制御回路を作ってみよう。エアチェックタイマ以外にも機器制御などでいろいろ応用ができるはずだ。

■受信料払ってますか？

大家，好久不見，イ尔們身体好口馬？

というわけで，なぜか中国語を勉強しています(例によって，漢字が正確でないけど許してね)。英会話の教材は，あふれるほど世にあるのに，なぜか中国語は少ないですね。「家出のドリッピー」中国語版とかあればいいのになぁ，と思いつつも，相対的に，有効なのがNHKのテレビ/ラジオ中国語講座です(寒い掛け合い漫談を我慢すれば)。

テレビはビデオ録画すればよいとして(1)，問題はラジオです。本放送が，8:20-8:40で，再放送が23:35-23:55に放送されており，朝は勤務中，夜は就寝中なのでテープに録音しなければなりません。

18:35 -「基礎英語1」
18:40 -「基礎英語2」
18:55 -「基礎英語3」
19:10 -「英会話入門」
19:25 -「英会話」
20:20 -「英会話」(再放送？)
20:40 -「やさしいビジネス英語」

と英語講座のあおりで，マイナーな中国語は深夜番組帯に追いやられています。

ひとりでぶーたれていても，しょうがないので，そういえば自分のラジカセにもタイマくらいついてたよなと思いながら，説明書を読んでみると，おやすみタイマ(CDとかテープの再生が終わると自動的に電源が切れる。寝ながら音楽を聴くときに便利)とおはようタイマ(ある時間になると音楽が鳴る，目覚まし時計代わりに便利)しかありません。まさか，と思って，近くの家電量販店へ行ってみても，だいたい最近のラジカセはこの2つのタイマしかないみたいです。

さらに数万円追加して，ミニコンポを買えば予約録音機能があるらしいですが，中国語講座を録音するためにミニコンポは買えません。そんなお金があったら駅前留学してますし，だいたい，置く場所もありません。

幸い，私のラジカセはテープ操作がボタン式で，録音状態にしておけば，AC100を入れて録音開始，切って停止を繰り返せます。ならば，外付けタイマでもいいや，と思って探したのですがアンティークなものしか見つかりません。録音を始めたい場所に赤い小棒を指して，終わる時間に白い小棒を指して……という機械的タイマばかりです。しかも，この小棒を差し込める穴が15分おきにしか開いていません。中国語講座は毎日20分なので，後半5分を無視するか，10分「名曲の小箱」を聞かなければなりません。

もう少し，粘って探したのが写真1です。これは，時間を指定するのに，小棒を差す代わりにダイヤルをあわせるというもので，これなら20分の番組も録音できそうです。

しかし，実際に使ってみると，ジージーと時計の動作音がします。昼間は，気になる音ではないのですが，夜，周りが静かになると，わりと気になる音です。また，しばらく使ってみると，どうも時間が少しずつずれるようです。

どうしようかなぁと考えているところへ，高尾氏がなにやらPICを使って，活動している模様。ないものは作れ，というOh!Xの精神にのっとり，エアチェックタイマを作りましたので，紹介します。

なんでもかんでも詰め込もうとすると，ビデオの予約録画にありがちな，操作性の悪いものになってしまいますから，今回は，中国語会話録音用タイマとして，以下の機能を中心に実現しました。

- 通常の時計としての機能
- ボタンにより，現在の時間，予約時間を設定可能
- 時間がきたら，AC100[V]の電源オン。一定時間経過後に，電源オフ
- 録音時間の指定は「分」単位。時計は「秒」単位
- 曜日の指定も可能

プログラムは付録のCD-ROMに入っているはずなので，もし足りない機能があれば各自で拡張してください。

たとえば，今回紹介するプログラムは汎用性を考えて，ある特定の曜日にのみ動作するようになっていますが，月～水の間，動作するようにすれば，より実用的かもしれません。また，PICマイコンのI/Oポートが3本余っていますので，月～水はラジカセAで入門編だけを録音，木・金はラジカセBで応用編を録音というようなより高度な動作も簡単に実現できます(ただし，操作が難しくなってしまうので，今回は実現していません。このようなデータは使用時に設定するよりも，プログラム中に埋め込んでしまったほうがよいと思っています)。

(1)先日，たまたま発見したのですが，ハングル語会話は，もっと寒いみたいです。「ハングル語体操」を見てしまって以来，母音の発音が頭に刻み込まれてしまったので，絶大な効果だったといえなくもありませんが……。

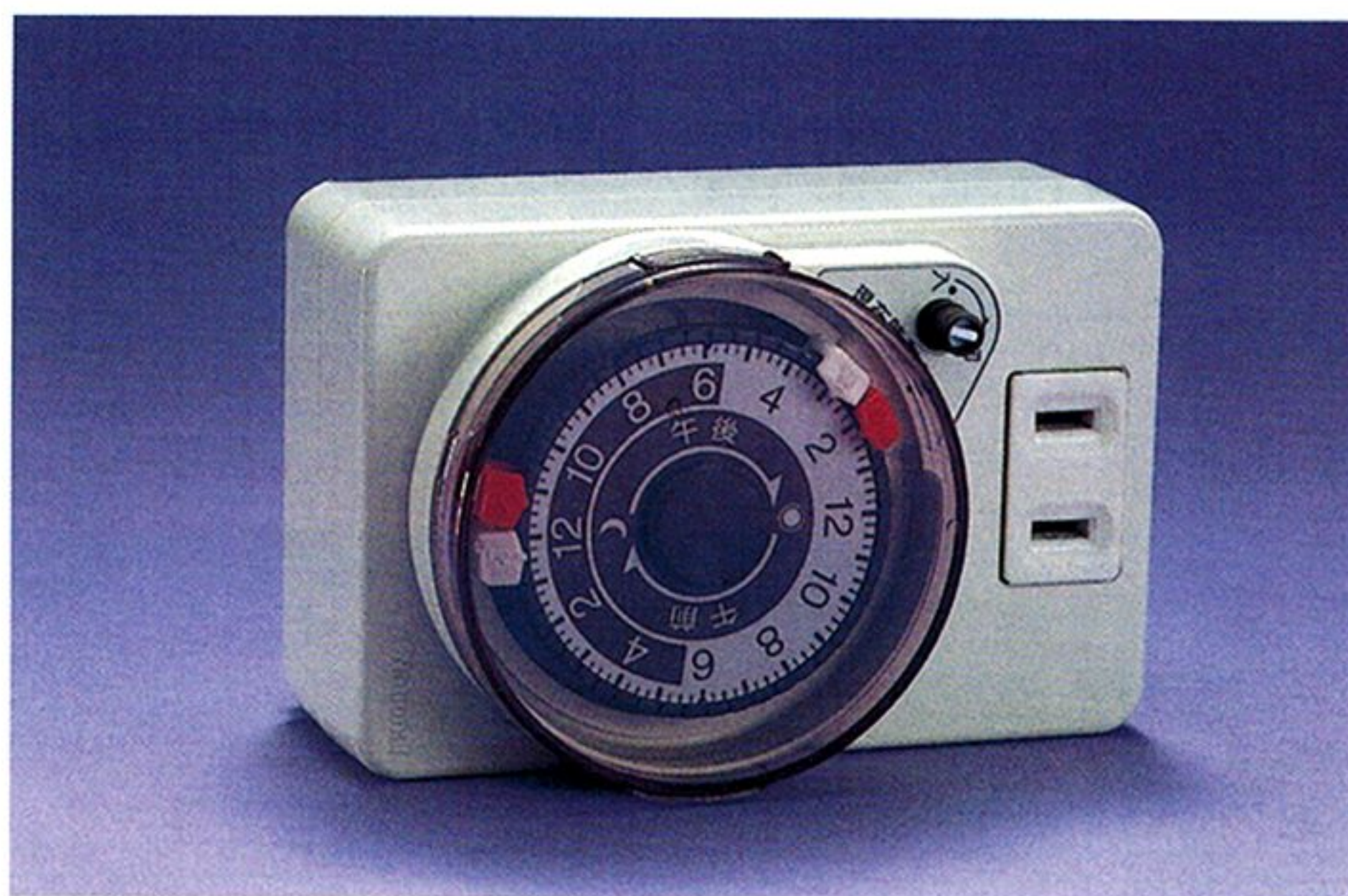



写真1 ダイヤル式タイマ(松下電工(株)TE331)

■クロック

さて，大まかな仕様が決まったところで，機能をブロックごとに見ていきます。

外部抵抗とコンデンサだけで自己発振する回路をPIC16F84は持っていますが，この周波数はあまり正確ではありません。時計を作るためには，正確な外部クロックが必要です。

電源には，写真2にあるように単3乾電池×4を使用しました。エアチェックタイマですので，AC100Vは，すぐ隣にあるのですが，待機時電圧を抑えようと思ったので電池を使用しました。このため，停電やコンセントを誤って抜いてしまった場合でも時刻が消えません。

電池駆動ですので，できるだけ消費電力を抑えなければなりません。いくら待機時電力を抑えても，毎月電池を買わなければならないのでは意味がありません。PIC16F84はCMOSのデバイスですので，動作周波数が低ければ低いほど消費電力も低いはずです。PIC16F84は最高20MHzまで動作可能ですが，単なるエアチェックタイマにそこまでの性能は必要ないので，動作クロックをできるだけ下げます。ただし，動作クロックを下げすぎて，時計の文字を1文字ごとにゆっくり書かれたり，ボタンの動作が我慢できないくらい遅くならないようにします。

以上の理由から，本セットでは32.768kHzの水晶を用いました。これは，主に腕時計やノートパソコン，携帯電話のRTC(Real Time Clock)などに大量に使用されているもので，安価ですし入手も問題ないでしょう。

PICマイコンは，4クロックで1命令を実行しますから，

32768 / 4 = 8192

で通常の動作時には1秒間に8192命令実行できます。

ただし，時刻設定時に文字の描画・消去を1/4秒ごとに行っていますので，これが最小時間処理単位になります。この場合，

8192 / 4 = 2048

となり，1秒間に2048命令実行できることなり，特に問題はなさそうです。逆に，これ以上遅くしても消費電力は下がらないでしょうし，使える命令数が減ってしまいますので，この値を用いました。

■LCDモジュール

本セットでは，時計の画面表示にSC1602BSLBというLCDモジュールを使用しました。このLCDモジュールとPIC16F84がセットになって「シリアル制御液晶モジュールキット」として，秋葉原の秋月電子通商で，見つけたからです。このキットは，PCのRS-232C端子につなぐと文字列が液晶に表示されるという基本的なものですが，PIC16F84とのインタフェイスがあらかじめ実装されています。PICマイコンに書き込まれたプログラムもソースリストが同封されていますので，解析も簡単です。さらに，

・5V単一電源動作
・8ビットあるいは4ビットパラレルインタフェイス可能
・16×2文字表示
・低消費電力(0.35 μA)

と，よいことずくめです。

ここ1年くらいは，秋月電子通商で安定的に供給されているようですし，ここは通信販売も受け付けているので入手性は問題ないと思います。

■ソリッドステートリレー

PICマイコンやLCDモジュールは乾電池の5Vで動作します。ラジカセも乾電池で動作しないこともないですが，大電力が必要ですのであまり経済的ではありません。今回はコンセントからのAC100Vで駆動します。

そういうわけで，5Vの制御信号で，100V電力を制御するのがソリッドステートリレー(Solid State Relay，略してSSRと呼ぶ人もいます)です。

機械式のリレー(スイッチを電気磁石で制御する)だと，AC電源をつないでいる間は，制御用の電気磁石を動作させなければならないので，消費電力がとても大きくなってしまいます。また，この電気磁石はON/OFF時に大きなノイズを発生し，マイコンを誤動作させてしまう可能性もあります。

それに対し，今回使用したシャープ製のS112S01シリーズは数mAで制御できますので，消費電力を抑えることもできますし，なによりもPIC16F84から直接制御することが可能です。

■プログラムの説明

プログラムには，第4章で高尾氏が発表されたオリジナルアセンブラを用いましたので，解析は容易だと思います。

大まかな流れとして，

(1) PIC16F84のペリフェラル/LCDモジュールを初期化
(2) タイマ割り込みを1秒ごとに設定
(3) 無限ループ。ただし，タイマ割り込みが発生したというフラグがセットされたら，1秒進めて，LCDを再描画
(4) 内部カレンダが，設定された時間になっていたら，SSRをONあるいはOFFする
(5) (3)へ

という感じになります。

タイマ割り込み処理ルーチンでカレンダーの更新とLCDの再描画を行わなかったのは，PIC16F84が多重割り込みをサポートしていないからです。本セットは，基本的な時刻表示のほかにも，ボタンが押されたら，時刻設定モードに入らなければなりません。このボタンが押されたことを検出するためにも割り込みを用いています。ボタンを押す人から見れば，PICマイコンが割り込み処理中か否かを選んで，ボタンを押すことはできませんから，運が悪いと，この2つの割り込みが重なることも考えられます。

(a) 内部カレンダー更新/LCD再描画中
(b) ボタンが押された
(c) 内部カレンダー更新/LCD再描画完了
(d) ボタン処理の割り込みルーチンへ
(e) 変更内容を表示するため，再びLCD描画
(f) 内部カレンダー更新/LCD再描画(その2)

最初はこのようなプログラムを組んでみたのですが，スイッチのチャタリングがひどく，一度スイッチが押されたら，次のスイッチを受け付けるまでにしばらく待たなければなりません(コンデンサをスイッチ出力とGNDの間に入れれば少しは改善されそうですが，なるべくシンプルに作りたかったのでやめました)。

時刻設定を行う場合は，数字を0.25秒間隔で点滅させますので，a)とf)の間で行える処理は，前述のように2048命令です。チャタリングのキャンセル用に空ループを回すには少し厳しい制限です。

そこで，以下のようなプログラム構成にしました。

・割り込み処理ルーチンでは，実際の処理(内部カレンダの更新，フォーカスの移動など)を行わずに，ただ，処理が要求されたというフラグをセットする
・1秒ごとあるいは0.25秒ごとのタイマ割り込みで，すべての要求されている(フラグのセットされている)処理を行う。
・必要な処理が終わったら，対応するビットをクリアし，次の要求が受け付けられるようにする。

実際のフラグは，表1のようにeventMaskというRAMに格納されています。

各割り込み処理ルーチンで直接処理せず，このようにフラグ経由で処理することで，重複する処理を避けることが簡単にできるようになります。前述の(a)～(f)の処理は，

(a') 内部カレンダー更新/LCD再描画中
(b') ボタンが押された(→フラグをセット)
(c') 内部カレンダー更新/LCD再描画完
(f') 内部カレンダー更新/LCD再描画(その2)
(d') ボタン処理ルーチンへ

と，(e)の処理を省くことができ，不必要にLCDが再描画されなくなります。

■データの格納

第2章でも説明されていますが，PICマイコンにはプログラム中にデータ列を入れることができません。命令の引数として定数を指定することはできますが，たとえば，

```
dc    "This is String."
```

とデータ列をプログラムメモリ内に用意して，ポ

インタで操作するという作業ができません。もし，データ列をポインタで扱いたければ，データRAM領域かEEPROMに格納しなければなりません。

本セットの場合，曜日名がデータとして必要です。

日曜日は"SUN"，月曜日は"MON"と表示しますが，これらのデータはプログラムに埋め込まれているのではなく，EEPROM内に格納されています(ラベルdayStringTable)。

レジスタeeadrに，その中でも表示したい文字列のアドレスが格納されているとして，以下のように表示します。

```
    w = 3
    cn = w
DisplayDay1:
    bank     1
    set eecon1,0 ;Initiate Read Cycle
    bank     0

    w = eedata
    eeadr++
    call     PutChar
    djnz     cn, DisplayDay1
    call     PutSpace
    return
```

なぜ，ここでBank切り替えや，eecon1のビット0をセットしなければならないのかはPIC16F84のデータシートのEEPROMの章を参照してください。

■文字の表示

使用したLCDモジュールには，点滅機能がないので，PICマイコンが文字の表示と消去を繰り返して，点滅を実現しています。これは，表示しようとする数値のビット7が，

0：通常動作

1：空白文字(" ")を表示(つまり文字を消去)

と表示ルーチン(時刻：DisplayDecimal，曜日：DisplayDay)で振り分けています。

タイマでセットされた時刻に達したか，「分」が繰り上がって「時」をインクリメントするべきかなどの判断時には，このビット7の情報が邪魔になります。このようなときには，サブルーチンUnmaskAllDataでこれらのフラグを一括してクリアします。

■部品

低損失3端子レギュレータ，LCDモジュールなど，一部，秋月電子通商以外では入手が難しい部品がありますが，それらさえ，なんとか入手できれば，あとは標準的な部品ばかりだと思います。

東京近郊にお住まいの方で，秋月電子通商に行かれる方は，土曜日の午後は避けたほうがいいでしょう。ものすごい込み具合です。このお店は，買い物客でごった返すというより，面白いものを探しに来ている人で込んでいるので，回転はあまり速くないです。以前はお店にしかない情報がたくさんあったのですが，最近はインターネットのホームページ(http://www.tomakomai.or.jp/akizuki)が頻繁にアップデートされているので，ほとんどの商品を通信販売で購入することもできます。

S8130は，5Vを安定して取り出すデバイスですが，入手できなければ，普通の3端子レギュレータでもかまいません。ただし，普通の3端子レギュレータは損失ドロップが大きいので，安定した5Vを供給するには，乾電池の6Vでは電圧不足です。単3×6本で9Vを作るか，乾電池駆動をあきらめてDCアダプタを使用するかしてください。SSRにAC100Vがきているので，そこから5Vを作成するのもいいかもしれません。

プログラムデバッグ中，私もDCアダプタを使用していたのですが，発熱はほとんどありませんでしたので，放熱板は不要だと思います。

LCDモジュールは，M1632とピン配置，コマンドともに互換性があるそうですから，同じものが入手できない場合は，M1632と互換性があるものを探してください(と，書いていていうのもなんですが，私はM1632を見たことはありません)。それも入手できないようなら，プログラムを改造するしかありません。

SSRは，PIC16F84のドライブ能力(25mA)で駆動できて，必要なAC電源を制御できるものならなんでもかまいません。今回は入手できませんでしたが，同じパッケージで，ゼロクロススイッチ(2)機能付きのS112S02を使用すれば，電源ON/OFF時のノイズをより低減することができます。

PIC16F84は，最近人気の出てきたデバイスですので，秋葉原・日本橋の電気街のパーツショップへ行けば入手できると思います。買ったままではデバイス内にプログラムが書かれていないので，付録のCD-ROMから，¥pic¥timer.hexを以下のオプションで書き込んでください。

Oscillator Option	LP
Watch Dog	Disabled
Power-up Timer	Enabled
Protection	No

(もし，書き込みプログラムにConfiguration BitのHex表示機能があれば0x10になっているはずです)

(2)単にAC電源をON/OFFさせるのではなく，AC(交流)電源が0VになるタイミングにIC同期させてON/OFFすることで，ノイズ，逆起電力を抑えることができる。逆起電力において，SSRはdtが小さいので，特に電源負荷がモータ，コイルなど大きなインダクタンス成分を持つ場合には重要。

■製作&動作確認

ほとんどの処理をPIC16F84で行っているため，回路構成は簡単です。配線はそんなに時間がかからないでしょう。私もOh!X復刊2号を見て驚いたのですが，全部カラーページですね。3号でもカラーページをいただけることになったので，読者の方が基板を作りやすいように，さまざまな色の配線材を使って2号機を作りましたので(写真2-1，2-2)ご利用ください。

配線が終わったら，PIC16F84，LCDモジュール，AC電源を差さずに，電池を入れてください。テスターがあれば，PIC16F84，LCDモジュールの電源ピンに5Vが供給されていることを確認してください。

電源を確認できたら，一度，電池を外し，PIC16F84，LCDモジュールを差して電池を入れてください。一呼吸おいて，LCDに，

SUN 00:00:00

と文字が表示されるはずです。文字が表示されない場合は，半固定抵抗を左右に回し，コントラストを調整してください。それでも表示されない場

ビット	ビットのラベル	対応する処理ルーチン	処理内容
0	ev_Timer	OnTimer	タイマ割り込み発生(LCDの再描画)
1	ev_ChangeFovus	OnChangeFocus	フォーカスの移動
2	ev_Adjust	OnAdjust	フォーカスされた数字をインクリメント
3	ev_Display1	OnDisplay1	LCDの1行目を再描画
4	ev_Display2	OnDisplay2	LCDの2行目を再描画
5	ev_Erase2	OnErase2	LCDの2行目を消去

表1 eventMaskの内訳

合は，以下の2つの理由が考えられます。

●PIC16F84が動作していない

→OSC1ピンの電圧を測る。矩形波で発振しているはずなので，5/2 = 2.5[V]近辺のはず。もし，0Vあるいは5Vなら，発振していないので，発振回路周りの配線あるいはPIC16F84のConfiguration Data（＝0x10）が正しく32kHzの水晶を使用できる設定になっているかを確認してください。

●PIC16F84とLCDモジュールのインタフェイスが間違っている

→写真2-3をよく見て，配線を再確認してください。

また，曜日の表示がおかしくなっている場合は，EEPROMへの書き込みが失敗しています。PIC16F84の書き込み機のマニュアルを読み，プログラムの開始アドレスが，アセンブラのデフォルト値と異なるようなら，明示的に指定し，アセンブルし直してください（秋月電子通商のオリジナルプログラムライタは異なるようです）。

ここまで動けば，基本的な回路に問題はないはずですので，適当な時刻をセットし，いよいよAC100[V]系をチェックします。現在時刻の1分後にON，その1分後にOFFと設定するとよいでしょう。

配線自体はとてもシンプルですが，なにせ100Vですから，注意深く試してください。初めは電気スタンドなど，負荷の小さいもので試し，SSRが熱くなっていないか，変な臭いがしないかを確認してください。

■拡張・改造するために

以上で，オリジナルエアチェックタイマを製作したわけですが，これでは単に動作の精確なタイマにすぎません。本セットの主な動作を制御しているのはPIC16F84というマイコンですから，プログラミング次第でもっと複雑な動作が簡単に実現できるはずです。デバイスのプログラム容量1K Wordに対し，今回のプログラムは約500 Wordしか使っていません。

時計機能を変更する必要性は少ないかもしれませんが，電源ON/OFFを変えることでより用途が広がるかもしれません。

SSRは，電源ON/OFFの機能しかありませんが，プログラムでより細かく制御して，ある一定の時刻だけ「点滅」するイルミネーションなども簡単に実現できるでしょう。もちろんある時刻だけ電源OFFするというような極性が逆のタイマを作成することも可能です。

また，本セットではひとつのSSRで，1本のAC電源しか制御しませんでしたが，PIC16F84には，あと何本かI/Oポートが余っているので，複数のAC電源を制御することも可能です。たとえば月～水曜日は基礎編だからカセット1に，木金曜日の応用編をカセット2にというような動作も可能です。

この場合，PICピン当たりの最大シンク電流は35mAですが，デバイス全体としての制限を超えないようにしてください。

実際に設定時間になった場合のAC電源ON/OFFは，サブルーチンControlPowerで行っているので，ここを変更すれば以上のような機能を実現できます。

プログラムの変更は，第3章，第4章の高尾氏の記事を見れば難しくないと思いますし，間違っても被害は少ないですが，AC100Vにはくれぐれも注意してください。今回は，ラジカセなど小電力のものを対象としましたが，炊飯器・クーラなどの大電力製品には絶対に使用しないでください。SSR自身は（適切な放熱板が正しくつけられたとして）12Armsまで扱えますが，そのためには，放熱以外にもスナバ回路，ZNRなどの外付け回路があったほうがよいですし，万が一のためヒューズをつけておいたほうがよいでしょう。また，ケースとしてプラスチックのものを使用しましたが，できれば不燃性のものを使用したほうがよいかもしれません。

本セットでは，そのような考慮があまりされてないので，用途は小電力アプリケーションのみにしてください。

参考にしたもの

(1)秋月電子通商「シリアル制御液晶モジュールキット」
(2)PIC16F84 Data Sheet, Microchip Technology
(3)S112S01 Series, S116S01 Series Data Sheet, Sharp

写真2-1　自作のタイマ

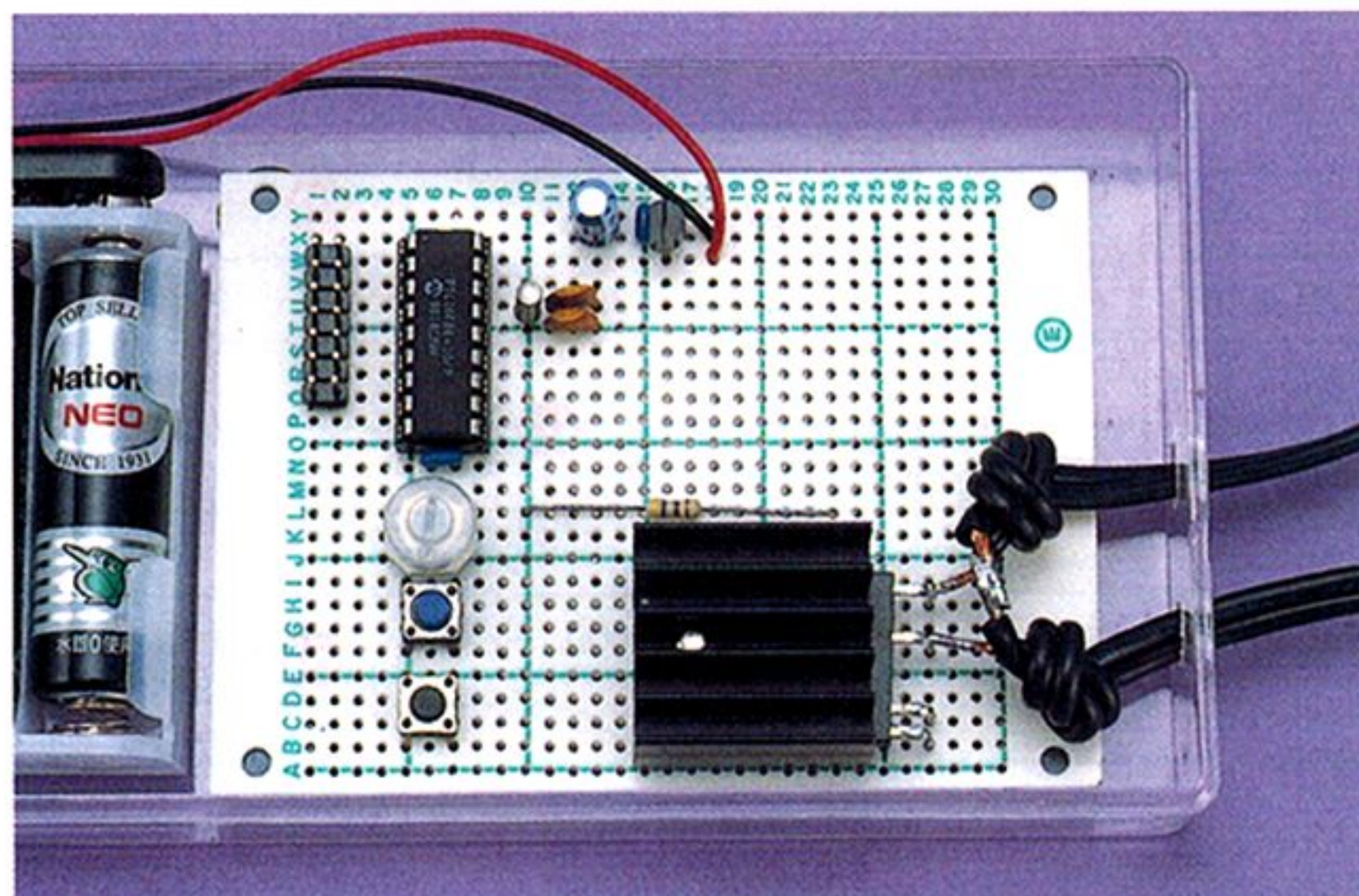
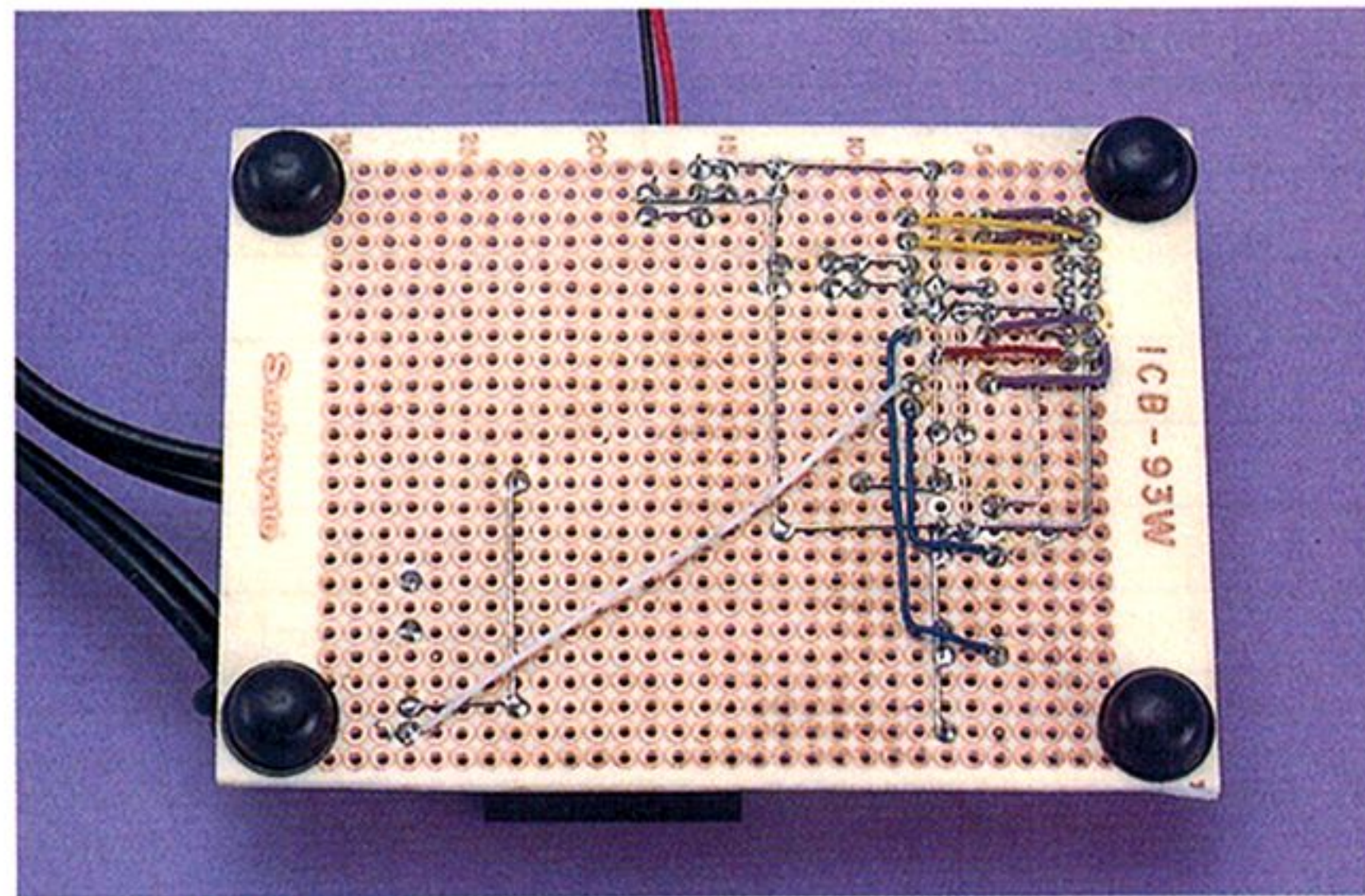



写真2-2　自作のタイマ（表）

写真2-3　自作のタイマ（裏）

Boxed Article

使い方

本セットには押しボタンスイッチが2つあります。

●フォーカス変更スイッチ

時間設定モード時、調整すべき値のフォーカスを変更する。

●インクリメントスイッチ

フォーカスされた曜日・数値をインクリメントする。

電源投入時は、通常動作を行っています。ここで、フォーカス変更スイッチを押すと、時刻設定モードになり、曜日表示が点滅します。インクリメントスイッチを押すと、SUN→MON→TUE..と曜日がインクリメントされます。フォーカス変更スイッチをさらに押せば、時間→分→秒とフォーカスが移りますので、それぞれの項目を現在時刻にあわせてください。

現在時刻設定後は、タイマ設定を行います。LCDモジュール2段目に

SUN 00:00+00:00

と表示されますので、ここで、AC100[V]ONする時刻と、ONする時間(長さ)を入力してください。

たとえば、火曜日のPM23:35-23:55のラジオ番組を録音する場合、

TUE 23:35+00:20

と設定します。次にフォーカス変更ボタンを押すと、

LCD >>>>>

という表示が現れますが、これは単に無視してください。これは、LCDの明るさ調節用に実装した機能なのですが、あまりにも使いづらいので、半固定で調節するようにしました。現在は未使用です(3)。

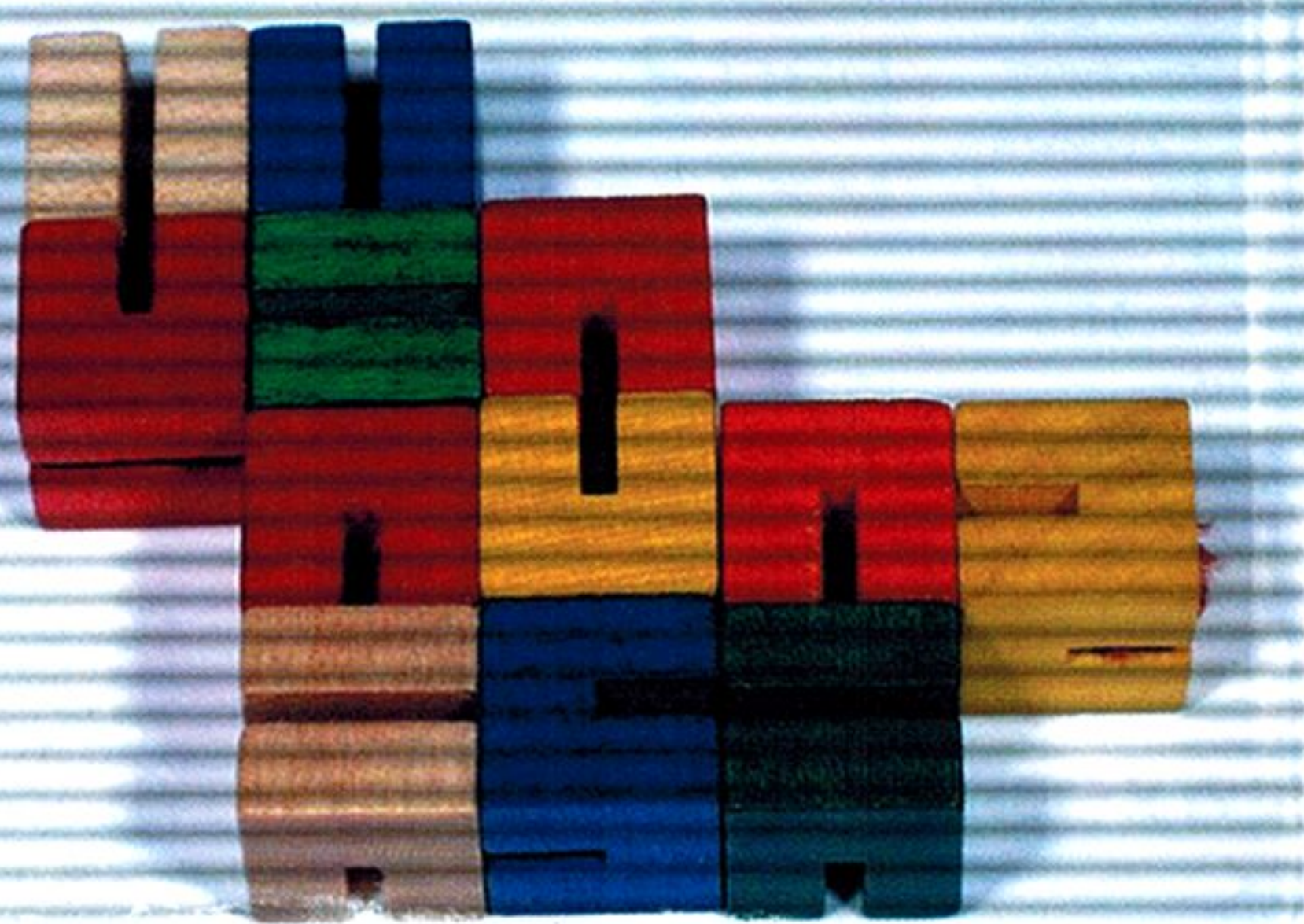
もう一度、フォーカス変更ボタンを押すと通常の動作に戻ります。

(3)本セットの消費電力のほとんどが、LCDの明るさ調節用の半固定抵抗です。ですから、ここをPIC16F84からPWM制御できれば、かなり消費電力を抑えられると思ったのですが、32.768kHz動作のPIC16F84では、PWMのパルス間隔が長すぎて、LCD表示がちらつくため、今回は断念しました。

部品名	型番	メーカー	数量	単価	購入場所
ワンチップマイコン	PIC16F84	Microchip Technology	1	600	秋月電子通商
LCDモジュール	SC1602BS	サンライト	1	900	秋月電子通商
低損失5[V]3端子レギュレータ	S8130		1	100	秋月電子通商
SSR	S112S01	Sharp	1	200	秋月電子通商
SSR用放熱板			1	100	千石電商
水晶発振子	32.768[kHz]		1	30	
抵抗	470Ω		1		
半固定抵抗	30KΩ		1		
セラミックコンデンサ	51pF		2		
	0.01μF		2		
電解コンデンサ	10μF		1		
ICソケット	18ピン		1		
押しボタンスイッチ			2		
ジャノメ基板	ICB-39	サンハヤト	1	200	
電池プラグ			1	50	
電池	単3		4		
電池ケース	単3x4用		1		
ケース	180x80x30[mm]		1	300	千石電商
コンセント(オス・メス)プラグ&ケーブル			1		

表2　部品表　その他に，PIC16F84にプログラムを書き込むプログラムライタが必要です。

特別企画

Oh!X流 デジカメ活用術

ここ数年間にわたりパソコンでもっとも人気のある周辺機器として君臨している
デジタルカメラ。その人気は留まることを知らない。同時に，デジタルカメラの
機能/性能も日進月歩の速い勢いで向上し続けている。
とはいえ，その進化の方向性には少し疑問はないだろうか？
QV-10によってデジカメは「カメラ」を超えた「デジタルなメディア」として登場
した。その用途は従来のカメラの枠では考えられない方向を示唆していたはずだ。
しかし，QV-10には欠点があった。画質が非常に悪い。その後のデジカメ業界は
カメラメーカーの主導するところとなり，アナログカメラの代替を目指して画質
の向上が競って行われることになった。
目標は明らかにアナログカメラである。
アナログカメラを目指しているわりには，操作性，拡張性などで大きく劣る面が
見受けられるものの，改善はなされず，もっぱら性能はCCDのスペックで表され
るようになってしまった。デジカメに関する記事はやたら不可解だった。デジカ
メライターと称する人々はなにを理解しているのか？
原色フィルタだとなぜ有利とされるのか？　正方画素が有利な理由は？　などな
ど疑問点をいろいろ聞いて回ったのだが，満足のいく回答は得られなかった。「デ
ジカメメーカーの人がそういってるから」というところにしか落ち着かない。
また，普通のカメラを意識するのはわかる。そこまでアナログカメラを指向する
なら，なぜもっと近づけないのか？　そんな疑問を抱いたことはないだろうか？
本来，デジタルメディアであったデジカメにはもっと違った進化の可能性があった
はずだ。現状までの展開を総括し，そのうえでなにが可能なのかを考えてみたい。

カメラを知らない人のための デジカメの基礎知識

荻窪　圭 Ogikubo Kei

デジカメも銀塩も「カメラ」であり，撮影原理の基本は変わらない。ここではカメラのもっとも基本となる用語や概念を解説して，どのような構造だからどのように扱うのがよいのかなどを概観してみよう。

どもども。荻窪圭である。たぶん，私はかなり初期の頃からパソコンとデジカメの「おいしい関係」に目をつけてそれを記事にした者のひとりである。かつてOh!X誌で連載していた「大人のためのX68000」で，「電子スチルカメラ」で撮った画像をビデオイメージユニットを通してX68000でキャプチャし利用する，って遊びを提案したのだ。このときの電子スチルカメラは「アナログカメラ」だったが，そこにはいつかデジタルデータとして画像を記録してパソコン上で直接利用できる世界がくるという確信があり，だからあたしはフィルムスキャナも買わず，じっと待っていたのである。

やがて，カメラの世界では富士写真フイルムが世界最初のデジタルカメラを開発し，パソコンの世界ではアップルが画像入力装置としてのデジカメである「QuickTake100」を世に送り出した。でもそれらは実用的なカメラとしての完成度はまだまだ低かった。

やがて，カシオがQV-10を出し，それが巷で評判となったことで，デジカメの急速な進化が始まったのだ。

そんなわけで，民生用のデジカメは「パソコン用画像入力装置のひとつ」として登場した。もっと正しくいえば「パソコン用画像入力装置のひとつとして普及を始めた」のである。だから，パソコンライターのあたしのところにデジカメのレビューがたくさん降ってきたし，いつのまにかデジカメの話を書く機会がどんどん増えてきた。その途中でふと気づいたのである。

「こいつはまぎれもなくカメラではないか」ってことだ。デジカメが画像入力装置の仲間からカメラの仲間へと世間での位置づけを替えようとするにしたがって，「カメラの世界」と「デジタル処理の世界」の両方の知識，概念が必要になったのだ。でも，パソコンの世界において「カメラについての知識や概念」はおろそかにされてきた。

そこで「カメラとしてのデジカメ」という観点からデジカメの基礎知識を積み上げるという行為を誰かがやんなきゃいけなくなったのだ。

でも誰もやってくれそうになかったので，あたしがやるのである。

図1　レンズと受光部の簡単な図

part1 とりあえず頭の中にカメラの基本原理を入れておく

カメラである。デジカメだろうが銀塩だろうがビデオカメラだろうがとりあえず全部「カメラ」だ。これらの基本動作原理はすべて一緒である。そんな話をしよう。

そこらじゅうを飛び交っている光のうち，レンズに入ってきたものが集められて受光部に到達し，それを画像として記録する。我々の目に映っているものはすべて「光」であるからして，当然のことだ。

人間の場合，水晶体を通った光が網膜上に像を結び，それが視神経を通って脳に伝わって「画像として解釈」される。

水晶体はカメラのレンズに相当する。

網膜は受光部である。銀塩カメラならフィルム，デジカメやビデオカメラならCCDだ。

「結んだ像を解釈する」のは銀塩カメラなら現像や焼き付けという過程であり，デジカメやビデオカメラだと内部で画像処理を施して記録メディアに記録する行為となる。

銀塩写真の場合，いったん記録メディアに記録してからそれを解釈する，という順番になるけれども，やっていることは概ね一緒だ。光をとらえ，それを解釈し，記録し，記録したものを鑑賞するわけである。

当たり前だけど，とりあえずこの順番で話を進めていこう。

part2 レンズとその周辺の話

カメラにはレンズがついている。人間の場合「水晶体」がひとつあるだけで，光学ズーム機能は持ってないし，レンズ交換もできない(はずである。いまのところは)。

カメラの場合，たいてい何枚ものレンズで構成されている。単焦点カメラでもフォーカスをあわせたり焦点距離を調整したり周辺の歪みをなくすためにいくつものレンズを組み合わせるのが普通だ。でも，その組み合わせを我々が意識する必要はない。大事なのは，レンズを組み合わせた結果，得られる性能だ。

レンズを通った光が集められて，屈折したり，なんやかやして，うまく受光部上に焦点があえば，きれいな写真が撮れるという仕組みなのだ。

・レンズと焦点距離

レンズには固有の焦点距離(俗にいう，レンズの長さ)がある。それが固定されていると単焦点レンズだし，可変ならズームレンズだ。焦点距離ってのは画角を決めるものである。広角レンズなら広い範囲が写るし，望遠レンズなら狭い範囲が写る。望遠レンズは遠くのものを撮れる，と思う人がいるだろうが，それは単なる結果だ。同じ大きさの枠に狭い範囲が写る＝その分でかく写る，ってことである。

つまるところ，レンズにおいて重要なのは画角なのだ。

画角は受光部(デジカメの場合はCCDだな)の大きさと焦点距離で決まる。焦点距離というのは受光部とレンズの中心(主点というんだけど)の距離のことだ。もうちょっと正しく書くと，無限遠にピントをあわせたときのレンズの光学的な中心と受光部の距離ってことである。ピントをあわせるために焦点距離が微妙にずれるから，基準は無

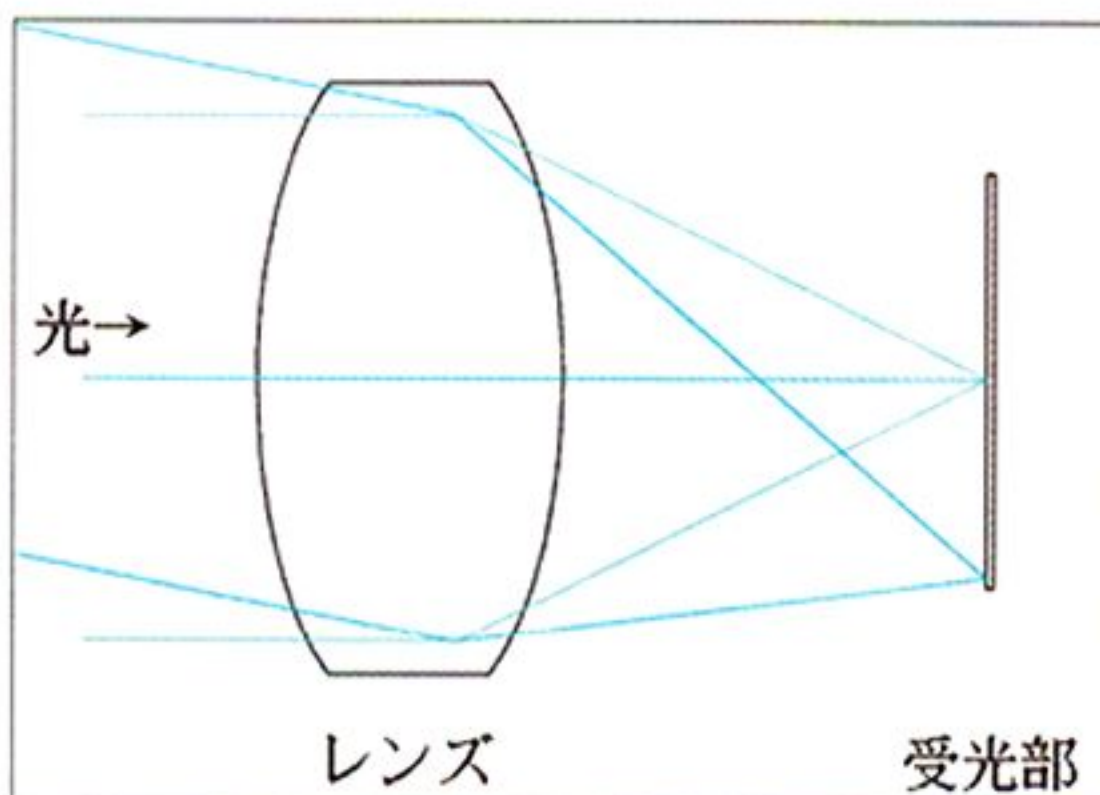
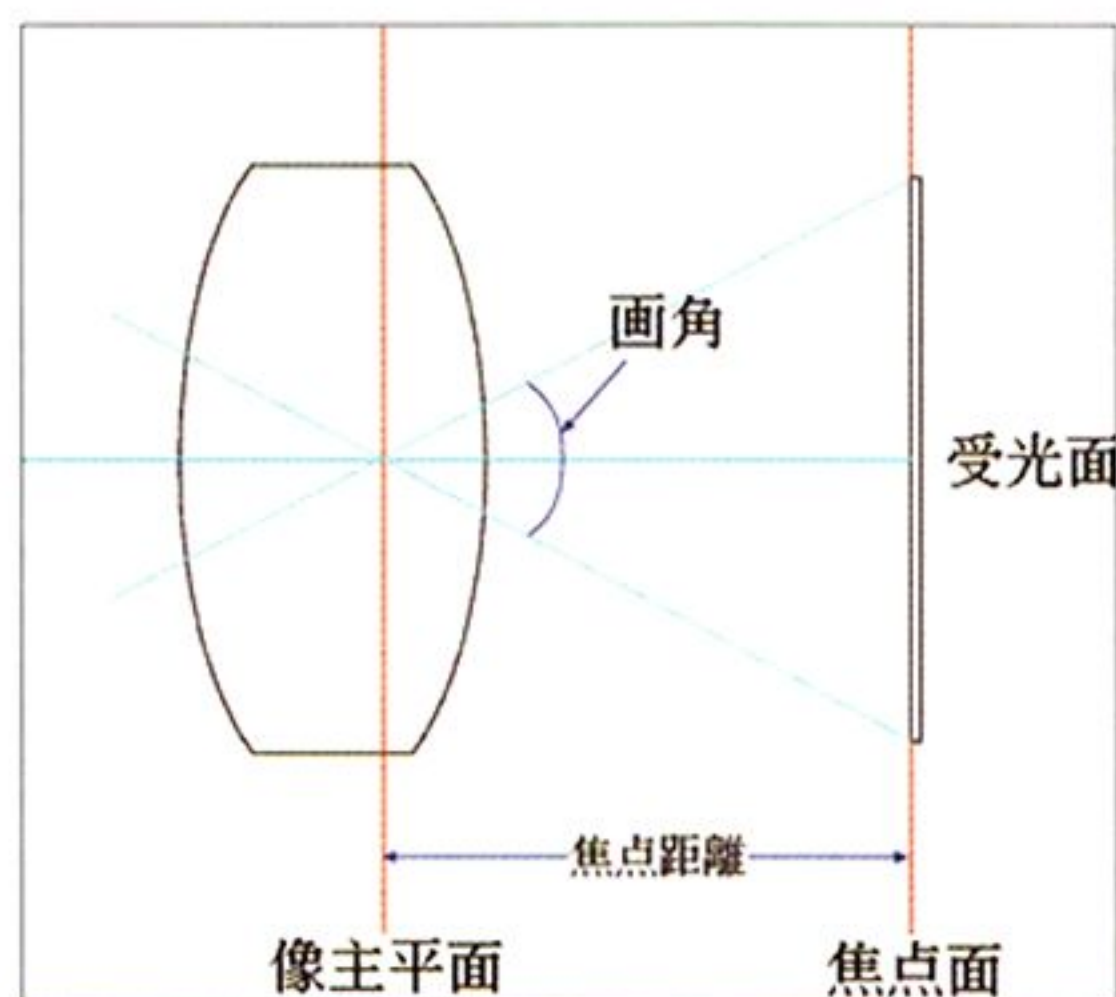



図2　焦点距離と画角の簡単な概念図

焦点距離(mm)	対角線画角(°)
14	114
24	84
28	75
35	63
50	46
70	34
100	24
135	18
200	12

表1　35mmカメラの場合の焦点距離と画角の表

図3　広角と望遠の作例

限遠にあわせたときってことになっているのだ。無限遠にピントが合うのは平行な光がレンズに入ってきたときである。

広角と望遠の違いはちょっと考えてみるとわかると思う。

レンズと受光部が近ければ広い範囲が写る＝画角が大きい＝広角であるし，レンズと受光部が遠ければ狭い範囲しか写らない＝画角が狭い＝望遠である，というわけだ。

でもカメラやレンズのスペックに「画角50°」って書くことはない。本当はそれがいちばん正確でいいと思うんだけど，実際にはレンズの焦点距離で書かれるのだ。35mmレンズとか50mmレンズという具合に。画角はこの焦点距離と受光部の大きさから勝手に計算してね，っていうのである。簡単な三角関数で計算できるからね。

それでは同じ焦点距離のレンズを使っても，35mmフィルムよりAPSのほうが受光面が狭いため，画角も狭くなるということになって，とてもややこしい。もっと受光面が狭いデジカメのCCDともなると，35mmフィルムカメラでは28mmなんて広角レンズだけれども，デジカメの28mmなんて超望遠なのだ。しかもデジカメのCCDはカメラによって大きさが違うため，焦点距離で記述されても画角がわからない。

そこでデジカメではたいてい本当の焦点距離と同時に，「35mmフィルム換算」の値で示される。このデジカメのレンズは焦点距離が6mmだけど，35mmフィルム換算だと38mm相当になるから，それから画角は想像してね，ってわけだ。

ちなみに，焦点距離が6mmだからといって，実際にCCDより6mmだけ離れたところにレンズがあるわけじゃない。それじゃああまりに近すぎて大変すぎる。そこで複数のレンズを組み合わせて，最終的に光学的には「6mm」という焦点距離に相当するように設計してるわけだ。

・画角の違いによる写りの違い

画角の広い/狭いは言葉どおり，広い範囲が写るか狭い範囲しか写らないかという違いである。でもそれに起因したさまざまな写りの違いというものが発生する。これが面白いところだ。

レンズが広角であるほど遠近感は強調され，望遠になるほど遠近感が圧縮されて写る。これは代表的な性質だ。それはどういうことかというと，そうだな，作例を見てほしい。

片方は約24mm相当の広角で，もう片方は約114mmの望遠で同じ被写体を撮ったものだ。メインの被写体が同じように写るよう撮影位置を変えてある。

両者の背景の写り方を見てもらいたい。明らかに異なっているのがわかるだろう。

画角の違いにはもうひとつ大きな特徴が表れる。「被写界深度」の違いである。広角のほうが被写界深度が深く，望遠のほうが被写界深度が浅い。「被写界深度」というのはどれだけ広い範囲までピントがあっているように見えるか，っていう指針だ。被写界深度についての詳しい話は別の項目で述べよう。けっこうややこしいから。

・レンズの明るさと絞りとシャッタースピード

「レンズ」の仕事とはいい切れない部分もあるけど，受光部に光が当たる前に行われる作業は全部ここに入れちゃうことにする。

レンズを通った光がCCDに当たることで画像が捉えられる，って話を最初にした。でもCCDも人の子だから，当たる光が強すぎると破綻しちゃうし，弱すぎると有効な信号を作れない。フィルムの場合，光が強すぎると真っ白にトンぢゃうし，弱すぎると真っ黒につぶれちゃう。人間の場合，光が強すぎるとまぶしくて目を開けてられないし，弱すぎると真っ暗でなにも見えなくて怖くて泣き出す。

よって，あまりに暗い場合はどうしようもないけれども，適度な光量があった場合は，それをCCDにほどよい強さに調整して当ててやらなければならない。それを担うのが絞りとシャッタースピードだ。

簡単にいえば，絞りで一度に流れ込む光の量を

■ 絞りとシャッタースピードの組み合わせ例

露出量が「EV9」の場合		露出量が「EV10」の場合		露出量が「EV11」の場合		露出量が「EV12」の場合	
絞り(F値)	シャッタースピード(秒)	絞り(F値)	シャッタースピード(秒)	絞り(F値)	シャッタースピード(秒)	絞り(F値)	シャッタースピード(秒)
2	1/125	2	1/250	2	1/500	2	1/1000
2.8	1/60	2.8	1/125	2.8	1/250	2.8	1/500
4	1/30	4	1/60	4	1/125	4	1/250
5.6	1/15	5.6	1/30	5.6	1/60	5.6	1/125
8	1/8	8	1/15	8	1/30	8	1/60
11	1/4	11	1/8	11	1/15	11	1/30
16	1/2	16	1/4	16	1/8	16	1/15

表2　絞りとシャッタースピードの表。EV値をベースに絞りとシャッタースピードを変化させてみた

注）多くのデジカメは絞り値を2，3パターンしか持たないため，その分シャッタースピードを細かくコントロールしている

調節し，シャッタースピードでその光がCCDに当たる時間を調節するのである。強い光が短い時間当たるか，弱い光が長時間当たるかって調整をするわけで，それを「露出」と呼ぶ。

水と同じだね。同じ量の水を溜めるのに，水が出てくる穴のでかさと蛇口をひねっている時間の組み合わせで決まるってことだ。

絞りというのは簡単にいえば，光を通す「穴」である。でかい穴だとたくさん光が通るし，小さい穴だと少ししか通らない。穴を小さくすることを「絞り込む」という。

絞りの単位は「F値」というので表す。ほとんどのデジカメのレンズには「F2.6」という感じで数値が書いてある。これはレンズの明るさを示すもので，絞りをいちばん開いたとき（「開放」という）の明るさ＝穴の大きさを示すものだ。これは口径比ってやつで決まる。

単純に考えてみよう。レンズの口径がでかければ（つまりレンズがでかければ），それだけたくさんの光を集めることができるのだ。よって，焦点距離と口径の比でレンズの明るさがわかる。実際のレンズは複雑なことになってるので「光学的な口径」がポイントになる。定規を持ってきてレンズ部の直径を測ってもダメだ。

で，F2.0というのは口径を1とした場合，焦点距離が2.0あるよってこと。ってことはオリンパスのC-2000ZOOMはレンズがF2.0の明るさで焦点距離が6.5mm（F2.0のときの焦点距離）なので，有効口径は3.25mmになるというわけだ。

それはいいとして，絞りはたいていの場合可変である。多くのデジカメは2，3パターンの絞り値しか持ってないが（例外はC-2000ZOOM），それはおそらく精度を上げるためにコストがかかるからだ。なにしろ開放時の有効口径が3mmって世界なのである。

で，絞り値は公比を2の平方根にした数列で示される。なんでかっていうと，絞り値は焦点距離と口径（直径）で示されるんだけど，実際に光が通るのは面積だから。半径が倍になれば円の面積はその二乗（つまり4倍）になるというわけ。

よって，F2.0のレンズはF2.8のレンズより2倍明るいのだ。2.0ってのは$1\times\sqrt{2}^2$であり，2.8ってのは$\sqrt{2}^3$の近似値だからである。F4.0のレンズはF2.0のレンズの1/4の明るさしかないのだ。

カメラの世界ではこの「2倍明るい」「2倍暗い」という単位をよく使い，それを「段」という。F2.0はF2.8より1段明るく，F4.0より2段明るいのである。この「段」という概念を覚えておくとカメラをいぢるとき非常に便利だ。

絞りに比べるとシャッタースピードは非常に明快だ。1/60秒，1/250秒と「秒」で表されるし，「1段」「2段」という感覚も，倍にしたり半分にしたりしていけばいいからだ。1/60は1/30より1段速いし，1/15は1/30より1段遅い。

・AEと露出補正

絞りとシャッタースピードをどうコントロールするか，というのが露出の大きなポイントだ。特にデジカメでは銀塩フィルム（特にネガフィルム）に比べてダイナミックレンジが狭い（要するに一度に対処できる輝度差が狭い）ため，露出が狂うと簡単に白くとんだり黒くツブれたりする。いくらPhotoshopで頑張っても写ってないものはしょうがない。

そこで露出の決定がなされる。ほとんどのデジカメはプログラムAEを使っている。AEのAはオートマティック，Eは露出で，要するに自動露出だ。で，必要な「露出量」（EVという）が決まると，あらかじめプログラムされたとおりにその露出に合わせた絞りとシャッタースピードの関係を決めるのである。

露出量を決定するためには「測光」を行う。光を測るわけやね。ここに各社の技術があって，オリンパスは「デジタルESP測光」，ニコンは「256分割マルチパターン測光」なんていってその正確性を競っている。事実，かなりいい感じである。リコーのように素直に「中央重点測光」を使っているカメラもある。いくつかの機種はスポット測光機能も持っている。スポット測光は中央の一部分だけを測光の対象にするもので，あまりに被写体とその周辺の明るさに違いがある場合なんかに役立つ。

で，その結果を基に露出が決められるわけだ。

それでも「どうもこの露出はおかしいんちゃうか」ってときは「逆光らしさを出すためにもっと絞りたい」とか逆に「青空は白くトばしてもいいから被写体を明るく」とか，なんらかの意図があるときは露出補正をかける。これは「＋1」とか「－1」っていう風に数字で表すけど，この単位は「段」である。＋1段だと1段分明るく撮影する（たとえば，シャッタースピードを半分にする），－1段だと1段分暗くする（たとえばシャッタースピードを倍に上げる）ってことが起きるわけである。

人物をある程度のアップで撮るとき，真っ白な服を着ているか真っ黒な服を着ているかで写りが違ってくることがある。黒い部分が多ければカメラは「暗いから露出量を上げよう」と判断しがちだし，逆に白い部分が多ければカメラは「明るいから露出量を下げよう」と判断しがちだからだ。

・オートフォーカスとマニュアルフォーカス

露出を決めただけでは写真は写らない。写らないことないけど，ちゃんと写るにはフォーカスも重要だ。フォーカスがあってないとボケボケのピンボケ写真になる。ボケた写真はいくらPhotoshopを使っても復元はできない。写ってないものはどうしようもないわけで，とても大事だ。逆にシャープに写っているものをボケさせるのはできないことはないんだけどね（情報量を減らす行為だから）。

フォーカスというのは焦点をピタッと受光面（像面という。ゾウメンっていうとどっかの麺料理みたいだけど，それはともかく，フィルム面とかCCD面のこと）にあわせること。

焦点距離ってのは無限遠にピントをあわせたときの値ってことを書いたと思うけど，それを微妙に動かすことで無限遠以外の位置にフォーカスをあわせることができる。

で，ほとんどのデジカメがオートフォーカスなんだけど，これがまたちょっと怪しい。オートフォーカスの方式にはいろいろあって，コンパクトカメラだと赤外線を発光して反射を見て距離を測る赤外線オートフォーカスなんてのがあったり（うまく働く距離が限られているのが難点），一眼レフカメラだとAF専用のセンサーを持っていてしかも暗いと補助光を出したりもするんだけれども，デジカメの場合，だいたい次のどれかである。位相差検出方式かコントラスト検出方式だ。

カシオのQV-5500SXや7000SX，（たぶん）コダックのDC-260も位相差検出方式を使っている。レンズの横に測距用のセンサーがついていて，位相差（要するに，ズレ）を見て距離を測るわけだ。

それ以外のほとんどのデジカメはコントラスト検出方式を使っている。ちょっとずつフォーカスをずらしながら実際にCCDが捉えた画像をチェックし，コントラストが最大になるような点で，ピントがあったと判断する方式である。

で，これは当然ながら「暗いとハズしやすい」。カメラによって違うので一概にはいえないが，たぶんたいていのカメラは中央部に重み付けをしてそこでコントラスト検出をするけど，それでうまくいかないときは画面全体を見る。そのとき，背景のほうがコントラストがはっきりしていたらそっちにピントがあっちゃうのである。たとえば「夜，バストアップの人物に夜景を入れてスローシンクロで撮りたい」というときにそれが起きやすい。背景のほうがコントラストが高いからである。カメラによっては，日中でも青空に浮かぶビル群な

図4　合焦についての概念図

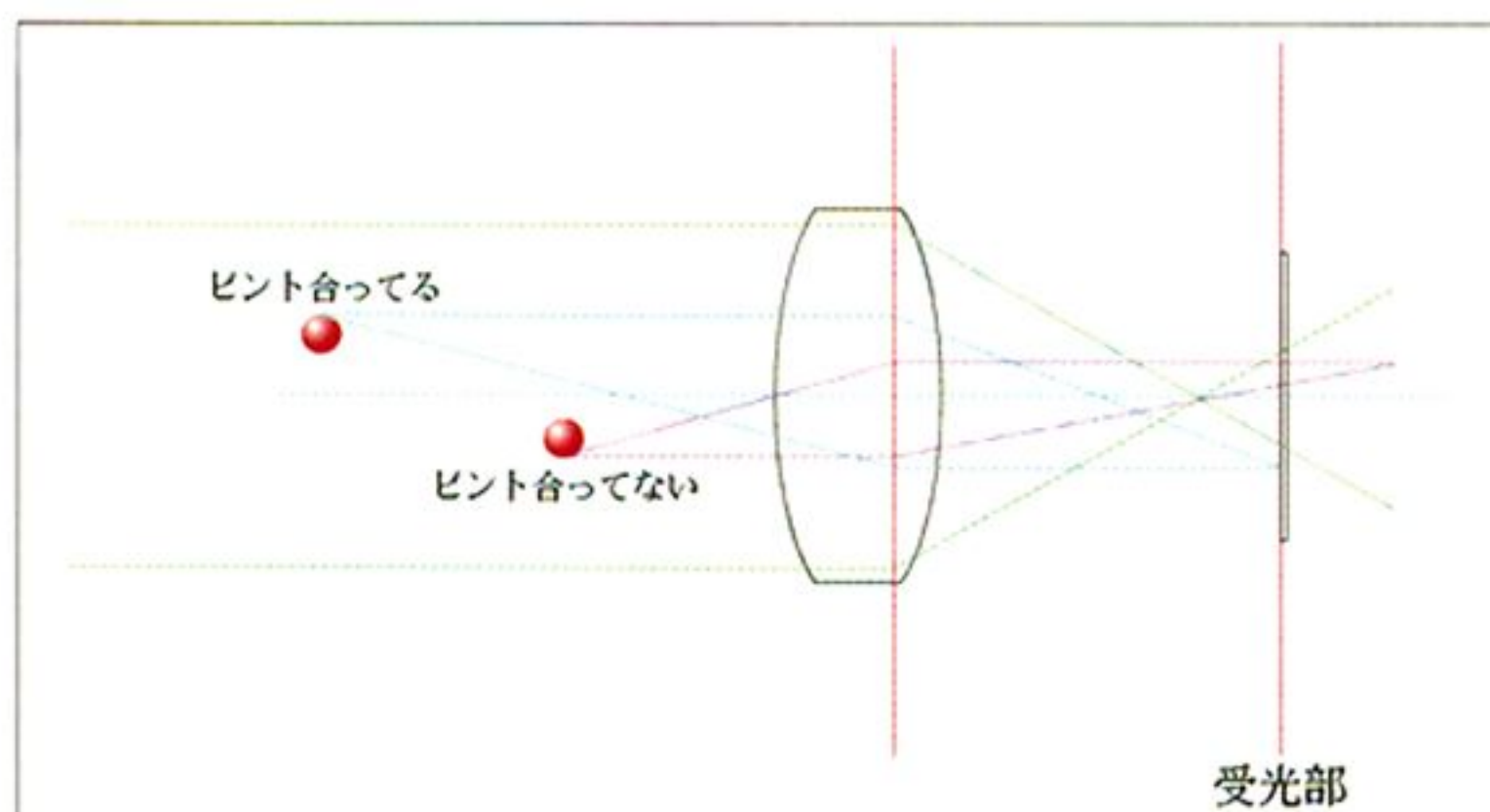
ピント合ってる
ピント合ってない
受光部

	35mmFilm				1/2インチCCD			
許容ボケ(mm)	0.035	0.035	0.035	0.035	0.008	0.008	0.008	0.008
絞り値(F number)	2	8	2	8	2	8	2	8
焦点距離(mm)	35	35	105	105	6.5	6.5	19.5	19.5
過焦点距離(m)	17.50	4.38	157.50	39.38	2.64	0.66	23.77	5.94
距離(m)	1	1	1	1	1	1	1	1
近点	0.95	0.81	0.99	0.98	0.73	0.40	0.96	0.86
遠点	1.06	1.30	1.01	1.03	1.61	-1.94	1.04	1.20
距離(m)	2	2	2	2	2	2	2	2
近点	1.79	1.37	1.97	1.90	1.14	0.50	1.84	1.50
遠点	2.26	3.68	2.03	2.11	8.24	-0.99	2.18	3.01
距離(m)	5	5	5	5	5	5	5	5
近点	3.89	2.33	4.85	4.44	1.73	0.58	4.13	2.72
遠点	7.00	-35.00	5.16	5.73	-5.60	-0.76	6.33	31.56
距離(m)	10	10	10	10	10	10	10	10
近点	6.36	3.04	9.40	7.97	2.09	0.62	7.04	3.73
遠点	23.33	-7.78	10.68	13.40	-3.59	-0.71	17.26	-14.64

表3　被写界深度

注）遠点が負の値になっているものは，無限遠まで被写界深度があることを表す。

んてのをバックにしたりすると人物を無視して後ろにピントがあいがちなものもある。

いやはや，ってなもんだ。で，マニュアルフォーカス機能を持つ機種もいくつかあるが，どれも液晶モニタを見ながらのフォーカシングになるため，暗かったり明るすぎたりするとピントの山(要するにピントがいちばんあう点)がわかりにくく，使いこなすのはしんどいのである。

だから最近のデジカメを使うときは注意するように。

・被写界深度の理屈と計算

さて，ピントを被写体のある点にあわせる，って話をした。では，そこ以外の点はピンボケなのか，っていうとそういうわけじゃない。ある程度許容範囲がある。「このくらいのボケならピントがあってるとみなしていいよん」という範囲だ。それが「被写界深度」だ。

たとえば，パンフォーカスのデジカメは，ある程度以上離れたものならどの距離のものを撮っても全部「被写界深度以内」だよん，って作りをしてある。実際には微妙に遠景が甘かったりするが，それはともかくとして，そうすれば，まんべんなくピントがあうので作るのは簡単だ。

被写界深度は焦点距離と受光面のサイズと絞り値と撮影距離(被写体＝フォーカスをあわせた位置との距離)の4つの組み合わせで決まり，次のような性質がある。

a. 焦点距離が短いほうが，つまり，望遠レンズより広角レンズのほうが広い範囲にピントがあう(ように見える)。

b. 受光面が小さければ小さいほど被写界深度は深くなる。

簡単にいえば，35mmフィルムは36×24mmの面積を持っているのに対して，対角線が1/2インチや1/3インチしかないデジカメは被写界深度が深い＝広い範囲にピントが合う，ってわけだ。

c. 絞り値が大きいほうが被写界深度は深くなる。F2.6で撮ったときより，F8.0で撮ったときのほうが広い範囲にピントがあうのである。

d. 撮影距離が短いほうが被写界深度は浅くなる。遠くのものを撮るときのほうが被写界深度は深くなる。

というわけで，面白そうなので実際に試算してみよう。

まず「許容ボケ」って概念がある。どのくらいのボケなら許すかってものだ。これを決めないと話が始まらない。一般に「八つ切りサイズにプリントしたものを25cmくらいの距離で見た場合，ボケて見えなければOK」ということになっているらしい。実際には八つ切りの場合「約0.2mm以下」ならよいそうだ。

キヤノンが発行している「EF Lens Works II」という本によると，キヤノンでは「フィルム画面対角線の長さの1/1000～1/1500程度」で，35mm一眼レフ用のEFレンズでは0.035mmを基準にしているそうな。これをベースに計算してみよう。

1/2インチのCCDの場合，対角線を1/2インチとすると対角線は約8mmである。その根拠は……1/2インチのCCDを採用しているリコーのRDC-5000用のパンフレットが手元にあるんだけれども，そこに1/2インチCCDの結像部分の縦横は6.4mm×4.8mmってちゃんと書いてあるのだ。ちょっとわざとらしい値だが，それを信じてみたわけ。

で，許容ボケを対角線の1/1000とすると0.008mm，1/1500とすると0.0053mmとなる。まあ，大きめにとって0.008mmとしよう。

計算すると表3のようになる。過焦点距離というのは「この距離まで離れたら，無限遠まで一応ピントがあうよ」って点のことだ。これを算出しておくとそのあとの計算が楽になるのである。

ちょいと表の見方を解説しておこう。35mmフィルムのカメラで35mmのレンズを絞り値F2で使った場合，17.5メートルより遠くにピントをあわせれば無限遠までちゃんと写りますよ，ってことだ。その下が距離による違い。被写体との距離が1mの場合，0.95mから1.06mの範囲にしかピントはあわないけど，10m離れたら6.36mから23.33mの範囲でピントがあいますよ，と。凄い違いでしょ。ついでに，F8に絞り込んで1mの距離の被写体を撮った場合，0.8mから1.3mまでピントがあいますよ，と。

これを1/2インチCCDのデジカメで考えてみると，35mm相当の画角は6.5mmのレンズで得られるので(ただ単に，C-2000ZOOMのカタログを見ただけである)，それで計算すると，たった2.64m離れただけでそれより遠くは無限遠までピントがあいますよ，と。10m離れたところにピントをあわせれば，なんと，0.62mから無限遠までピントがあいますよ，ってことなのである。

でも，105mm相当の望遠(19.5mm)側にすると，10メートル離れたものを撮った場合でも7～17mの間にしかピントがあわなくなるってのがわかる。

まあ，目安にすぎないけれども，こんな感じで被写界深度に大きな差が出るのだ。ピンボケを防いだり，手前の人物から背景まで全部にピントをあわせたいというときは，この表を見てどうすればいいか考えるとよい。それもひとつの技である。

だから銀塩写真と民生用デジカメで撮り比べた場合，解像力や発色はある程度がんばれたとしても，この被写界深度の違いだけはどうにもならないのである。よりボケの効いた写真が好きな人は，それなりに工夫しなければならないのだ。業務用デジカメならでかいCCDを使っているから銀塩フィルムに近い結果が出てくるんだけどね。たとえば，NikonのD1はAPS相当大きさを持つでか

いCCDを使っているため，かなり綺麗なボケが出せるはずだ。

part3 CCD

で，なんやかやと光量やらフォーカスやらを調整された光はCCDに当たっている。こいつが受光してそれを電気信号に変換し，画像の元を作り出すのである。

それがまた厄介なのだ。

原則として，CCD自体の性能は1画素のサイズで決まる。1/2インチで211万画素のCCDと1/2インチで150万画素のCCDがあった場合，後者のほうが1画素当たりの面積は大きい。面積が大きいってことはそれだけたくさんの光を受けることができる＝感度が高い可能性がある，ってことである。

銀塩フィルムの場合，入れるフィルムの感度によってそれをコントロールできる。ISO400のフィルムはISO100のフィルムの4倍の感度を持つってことだ。ISO100のフィルムでは1/60秒でしかシャッターを切れないときでも，ISO400なら1/250秒で切れるのである。

デジカメの場合，CCDを取り替えるわけにはいかないので，どんなCCDを使っているかは重要なのだ。だが，ここがまたややこしいところで，CCDの技術も進歩しており，できるだけ1つひとつの画素に無駄なく光を当てようって技術もまた進歩しているわけで，同じ画素サイズでも去年のCCDより今年のCCDのほうがより感度が上がっていたりするのだ。あくまでも原則は原則である。

ちなみに，現在デジカメ用のCCDを作っているメーカーはそう多くなく，代表的なのがソニー，松下電器，シャープの3社である。たとえば200万画素のデジカメのカタログのCCDの項目を見ると，211万画素，214万画素，230万画素と3種類あることがわかるはずだ。それぞれCCDのメーカーが違うと思っていい。ここを見れば「これとこれは同じCCDを使っているのかな？」っていう指針くらいにはなるだろう。

だったら，でかいCCDを使えば高画質のデジカメができるんやんけ，といわれればそのとおりである。たとえば1インチで200万画素のCCDなんて使えたら，すごく画質も感度も上がる。でも，CCDは半導体であり，ウエハーから切り出すものだから，でかければでかいほど歩留まりは悪くなり，ひとつのウエハーから切り出せる量も減るので，コストがグンと上がるのだ。かくして，業務用のデジカメは100万円，民生用のデジカメは10万円という差が出るのである。ニコンのD1が65万円もするのは，APSサイズの大きなCCDを採用しているせいでもあるのだ(よって，ニコンがD1の普及クラスを作ったとしても，同じCCDを使う限り，極端に値段が下がることはないはずだ)。

・CCDとカラーフィルタ

CCDの性能を決めるにはもうひとつポイントがある。「カラーフィルタ」である。

CCDは光の強さしか感じることができないので，そのままでは「モノクロの画像」しか撮れない。そこで，CCD上の各画素に「あなたは緑」「あなたは青」「あなたは緑」「あなたは赤」という風に色を担当させているのだ。それがカラーフィルタである。

で，ここで考えてみよう。

カラーフィルタをRGBの3色用意すると，実際の解像度は200万/3で約67万しかあらへんやんけっ。でも実際のデジカメは1600×1200＝192万ピクセルの画像作ってるやんか。詐欺ちゃうかっ。

まあ，詐欺といえば詐欺かもしれないんだが，いろいろとテクがあって。

たとえば，RGBの3色用意した場合，画素に対してRGBRGBRGBと順番に割り当てるってことはまずない。その辺は人間の目の特性をうまく利用するってことで，人間の目は色より輝度に敏感で，輝度信号はG(緑)信号に多く含まれているため，GRGBGRGBGRGBGRGBと「G」を多く配置していくのである。でもって，RやBの足りない分は，画像処理でもって「ここはきっとこの色だろう」って補間するのだ。かくして1600×1200ドットの画像ができあがる。では，もし補間なんてしないで200万画素のCCDで1024×768ドットの画像を作り出したらすげー綺麗なんじゃないだろか，そのとおりである。

ちなみに，

GRGBGRGBGRGB……
BGRGBGRGBGRG……
……

と並んでいるのをベイヤー配列という。

GRGBGRGBGRGB……
GBGRGBGRGBGR……
……

と並んでいるのをグリーンストライプ配列といったりする。

後者より前者のほうが画像の補間はしやすいが，技術的には後者のほうが作りやすいらしい。たとえば，補間に失敗すると，滑らかな線になるべきところが凸凹になったりする。

さてこれは「原色フィルタ」の場合。もうひとつ，「補色フィルタ」ってのがある。原色の補色であるCMYをフィルタとして使うものだ。CMYというのはシアン・マゼンタ・黄色のことで，「色の三原色」だ。原色フィルタより補色フィルタのほうが感度を上げやすい，解像度を上げやすいということがあって，最近では補色フィルタを使うデ

図5　補間がうまくいかなかった画像の例。DS-20にて撮影

図6　FinePix2900Zのシャープネスの作例。シャープネスオフから強までの4段階で撮影をし，中央部分を切り出したもの

図7 増感することでノイズが増えた例。CP-800にて基準感度，2倍，4倍の3パターンを撮影。影の部分を見るとノイズがどう載っているかがわかる

ジカメも多い。ニコンのCOOLPIX 950やオリンパスのC-2000ZOOMがそうだ。実際には，輝度信号が多く含まれる「G」は欠かせないのか「CMYG」の4色を使った配列が多い，と思う。そういうのはメーカーは教えてくれないので詳しくは知らないけど。

で，補色フィルタでも最終的にRGBにしなければならないわけで，「赤」を作るには「シアン」と「マゼンタ」を足してやらねばならないという事情があり，原則として色再現性では原色フィルタに劣るとされている。

どっちがいいかは難しい問題だけどね。原色フィルタを使っていても「なんか色がヘン」ってデジカメもあるし。理想的には感度の高い原色フィルタのCCDってことになりそうだけれども。

・絵を作る

補間をして色を作ったままの画像は細部が鮮明ではなく，写真としての「解像感」に欠ける。そこで，多くのデジカメはシャープネスの処理をかけ，シャキッとした絵を作り出す。

シャープネスを強めにかけると見栄えのする写真になるが，元の情報を失ったり，不自然さが出てしまったりする。たとえば，遠くから木々の緑を撮ったような「うじゃっ」とした絵はシャープネスをかけることで不自然になってしまう。

富士写真フイルムのFinePix2900ZはSETUPでシャープネスを「完全なオフ」から「最強」まで5段階用意している。最強は文字情報などを抽出したいとき，上から2番目は業務用としてカリッとして絵が求められるときのためで，下の3つが一般向けと思っていい。

その3つの違いをチェックしてみよう。

「オフ」は確かにピントが甘く見えるが，非常に自然な絵が得られているのがわかると思う。

図8 暗部を無理やり持ち上げてみた例。FinePix2700で撮影した夜景をベースに無理やり明るくしてみた

・増感してみよう

CCDから出てきた信号を増幅してやると，暗い＝弱い信号を明るくすることができる。「増感」である。最近はこの機能を持っているデジカメも増えてきた。増感してやればISO200とかISO400相当というCCDの性能以上のの感度が得られるわけだ。

でも，当然ながら増感すると一緒に「ノイズ」も増幅される。S/N比が悪くなるのである。デジカメによっては頑張ってノイズを抑えているものもあるが，やはり増感して撮ったものはそうでないものよりざらついた感じになるのは否めない。しかたがないところである。それでも撮りたいことってあるしね。

デジカメで暗いところを撮って，グラフィックソフトでレベルを思いっきり上げてみたり，彩度を思いっきり上げてみよう。そうすると，偽色がたくさん混じってるというのがわかるはずだ。

図9 C-1400XLの色温度を変えて撮影した例。3000Kから6500Kまでである。背景の白い紙の色を見ると変化がわかりやすい

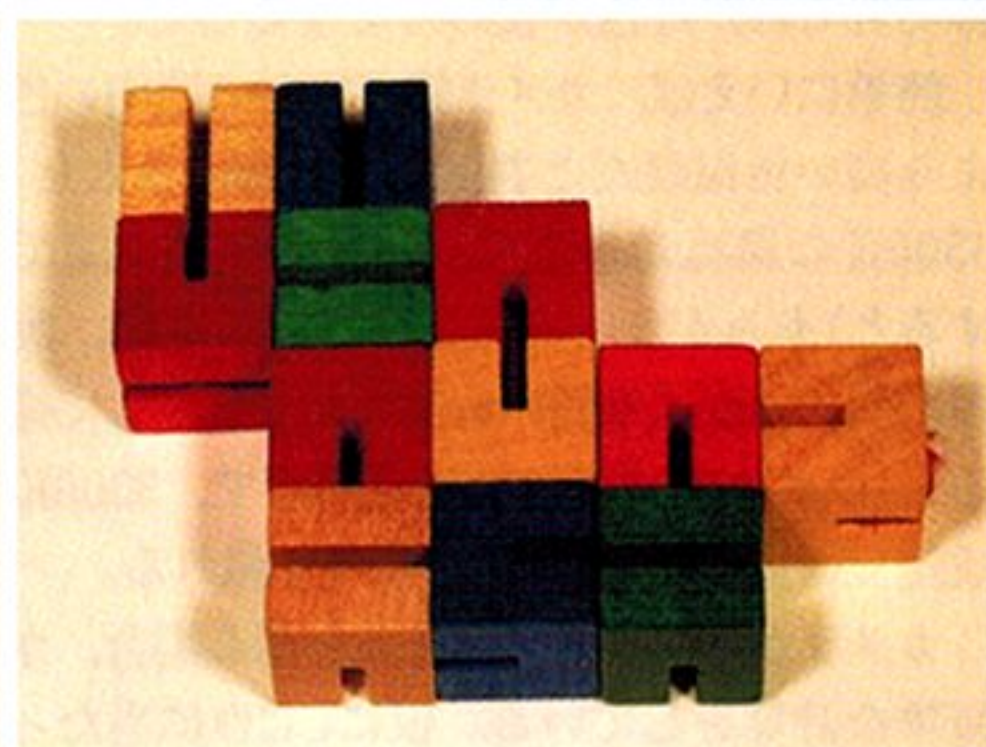
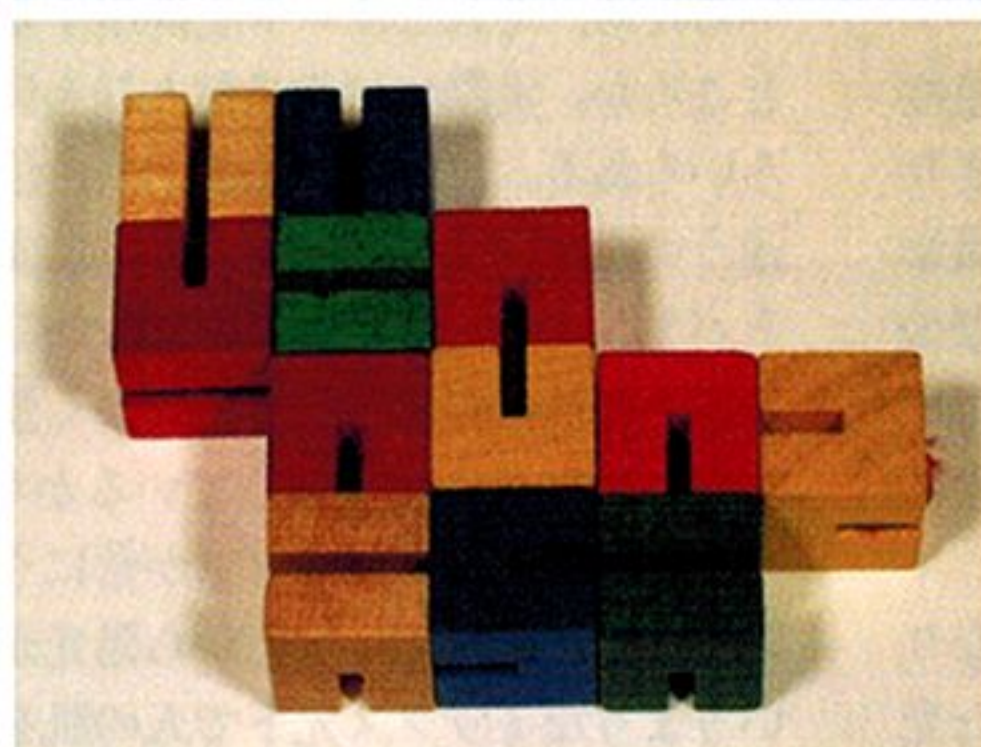
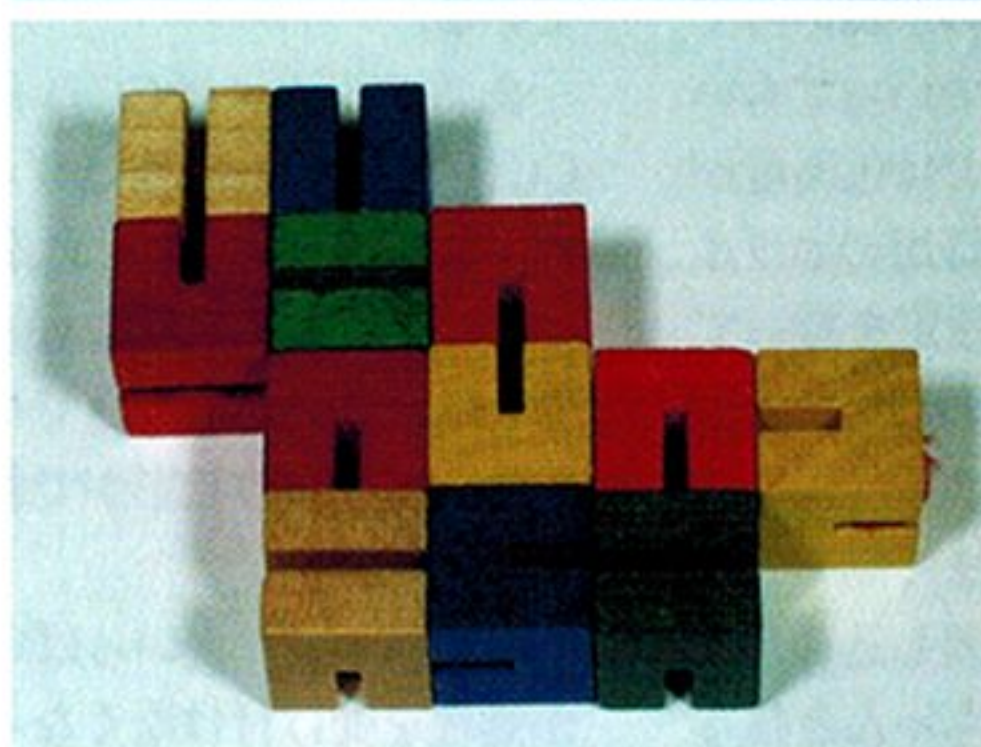
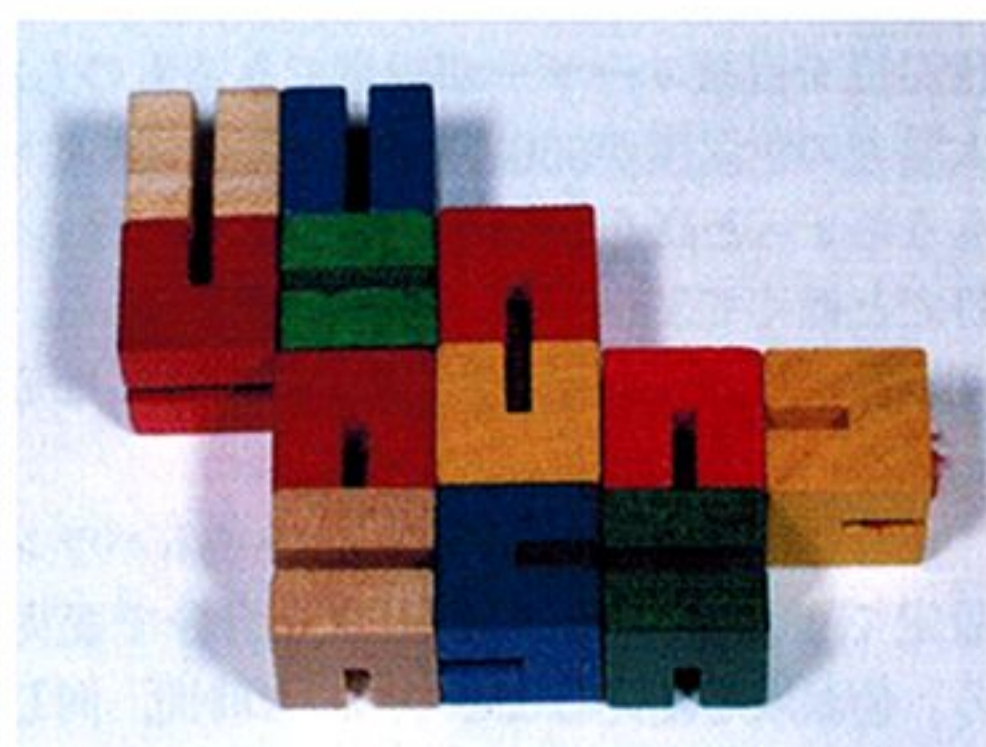
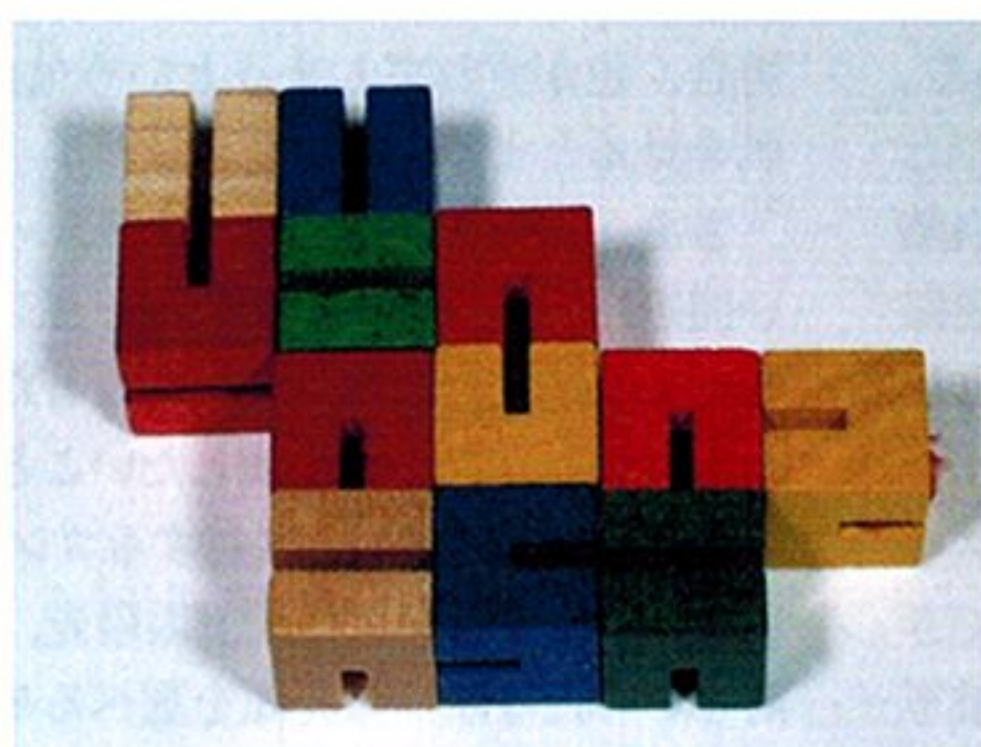
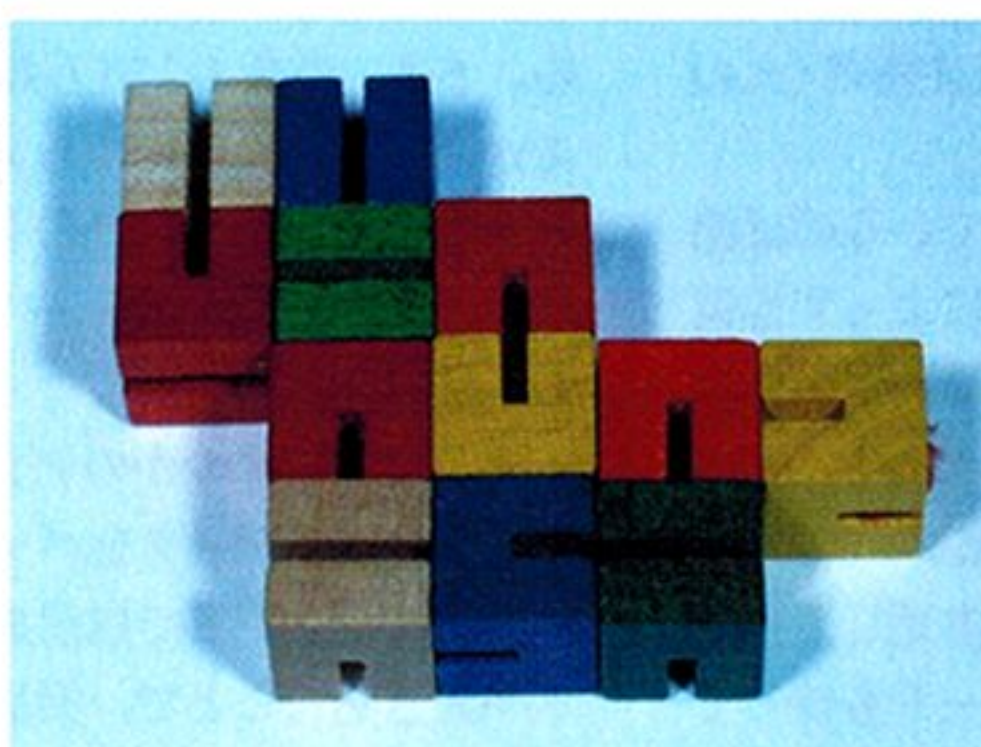



図10 プリセットホワイトバランスを使って色をコントロールした例。白に対してバランスを取る(これが基本)，緑色のものを白としてホワイトバランスを取る，赤いものを白としてホワイトバランスを取る，青いものを白としてホワイトバランスを取る。それぞれ極端に色がずれているのでわかりやすい

・ホワイトバランスを設定しよう

CCDから出てきた信号をもとにカラーの画像を作り出すわけだが，ここで「どれを何色と見なすか」という銀塩写真にはない問題が発生する。世の中の色ってものはおしなべて「光源の光を反射した結果」なんである(光源そのものの色ってのもあるが)。真っ白な光で白い紙を見ると紙は白いが，真っ赤な光の下で白い紙を見るとその紙は赤いのだ。当たり前のことなんだけどね。

銀塩写真の場合，光源にあったフィルムを使うことでそれに対処している。いちばんポピュラーなのは「デイライト用」，つまり昼間の太陽光を基準にしたフィルムで，これを使って蛍光灯下の人物をストロボ(フラッシュ)なしで撮るとヘンな色になる。白熱灯下の人物をフラッシュを焚かないで撮ると，赤っぽくなる。それがあまり知られていないのは，多くの人は「暗いところではフラッシュを焚く」からだ。

デジカメの場合，それをカメラ内で補正することが可能なので，照明条件が変わっても適正な色が得られるよう努力する。それが「ホワイトバランス」ってヤツだ。これがまた難しい。各社とも微妙に異なった方式を使っていて，微妙に異なった結果が得られたりするからである。

ホワイトバランスの設定をカメラ任せ(つまり，オート)にした場合を考えてみよう。

基準になるのは色温度だ。C-1400XLのように色温度を直接ユーザーが設定できるものもある。太陽光の色温度が5500K(ケルビン)。曇天下だともうちょっと高くて6500Kくらい(かな？)。白熱灯だと低くて3500K(くらいかな？)である。色温度が高いほど青白くなり，低いほど赤くなると思えばいい。光の波長の違いだ。

オリンパスのC-1400XLはホワイトバランスを設定できる代わりに，色温度を自分で変更できる。色温度を変えることで，同じ時間，同じ被写体でもこれだけ色が変わって写るのだ。

簡単にいえば，カメラ側が実際の照明の色温度より高い色温度だと判断して撮影すると(つまり3500Kの環境光の下で，5500K用の設定で撮影すると)赤みがかった写真が撮れ，逆に，実際の照明の色温度より低い色温度だと判断して撮影すると(つまり5500Kの環境光の下で，3500K用の設定で撮影すると)青みがかって撮れる。

カメラがホワイトバランスを測る場合，かなり複雑な計算をしている。仮にCCDに当たった光の中にあらゆる色が均等に含まれていて白トビもない場合，もっとも明るい部分を「真っ白」と判断して色を調整してやればいい。RGBの値がそれぞれ255・250・250だったとすれば，ちょっと赤みがかかっているというわけで，赤を抑えてやればバランスが取れるわけだ。でもこの方法だと，単色に近いものを撮ったとき(真っ赤なクルマをアップで撮るとか)に破綻する。よって，ある程度想定された環境のどれに近いかを判断したり，極端な照明環境はないと判断するアルゴリズムを入れたり，色の分布によって判断したりしているようだ。だからカメラ側が想定してない人工的な環境で撮影するとオートホワイトバランスがうまく働かないこともある。

富士写真フイルムの99年春夏モデルは「シーン自動認識型オートホワイトバランス」をうたった機能を搭載している。これは，あらかじめカメラの中に何パターンもの「環境光」のデータを持っておき，撮影した環境がそのどれに該当するかを判断して，それにあわせた補正をかけるという方式である。どれにも該当しない場合は太陽光時の補正をかけるようだ。これは確かに有効で，照明の雰囲気を残しつつ，自然な発色を実現している。

逆に，ニコンやオリンパスやエプソンやサンヨーのデジカメはそのときどきの判断でホワイトバランスを計算しているように見える。その証拠に，多少環境光が変わっても白いものをちゃんと白く描写しようとする。しかし，それが完璧ではない場合，色がずれてしまうという欠点がある(人の肌を見るとよくわかる)。

さあ，どちらがいいでしょう，っていうのは深い問題だ。これはシチュエーションによっても違ってくる。完全に補正してもらったほうがいい場合もあるし，雰囲気を残したいこともある。その辺はメーカーの絵作りに対するコンセプトの差だ。

多くのデジカメは曇天，太陽光，蛍光灯，白熱灯といったなかから光源を選べるモードも持っているため，それとオートを併用するのが賢いってところか。実際には蛍光灯とひと口でいってもいろいろある。白色，昼光色，昼白色では色温度が違うのだ。富士写真フイルムは蛍光灯用のホワイトバランスとして2つないし3つの設定を持っている。これは市販されていてよく使われている蛍光灯にあわせた設定にしているからだ。

オートホワイトバランスが常にうまく働くとは限らない。室内だけど窓から陽光が差し込む，というようなミックス光下で人の肌をちゃんと描写できるデジカメはそう多くはないのだ。

そこで最近はニコン，サンヨー，エプソンなどから「プリセット型のマニュアルホワイトバランス」を持つデジカメが増えてきた。これはあらかじめ撮影したい照明光の下で「真っ白なもの」を撮影して，それを「白」とみなすようなパラメータを計算してセットするのである。便宜的に真っ白なものといったが，具体的には「明るい色で無彩色のもの」と考えていいだろう。

これを利用すればほとんどの環境光下で適正な色補正が可能になる。逆に，彩度のある色のものを「白」と認識させることで，ある程度色補正をコントロールすることも可能になる。

part4
画像処理をして記録する

で，画像が作られると，次はメディアへの記録である。

多くの場合，RGB化したデータにJPEG圧縮をかけ，ファイルサイズを小さくしたうえで記録メディアへ書き込む。もしこの処理をシーケンシャルに行えば，1枚撮影するのに10秒近くかかってしまう。まあ，CPUの性能やらソフトウエアにもよるが，最近の200万画素デジカメでだいたい4～7秒が平均的なところだ。そこに画像処理→JPEG圧縮→書き込みの時間が含まれるのである。まあ，昨年の130万画素デジカメでは10秒近くかかっていたのだから大したものであるが，フィルムを使ったカメラの巻き上げにかかる時間に比べるとまだ長い(コンパクトカメラでもだいたい1～2秒である)。

そこで，いくつかのデジカメは内部にバッファを持ち，撮影データをバッファにため込みながら，バックグラウンドで以前撮った画像を記録メディアへ書き込むことで，撮影間隔を短くしようとしている。

あるデジカメはCCD-rawのデータ(要するにCCDの生データ)をまずバッファにためておき，バックグラウンドでそれを処理して圧縮して書き込みつつ，次の撮影にも備える，という技を使っている。こうすることでバッファがいっぱいにならない限り次の撮影ができる＝撮影間隔が短くなるわけだ。

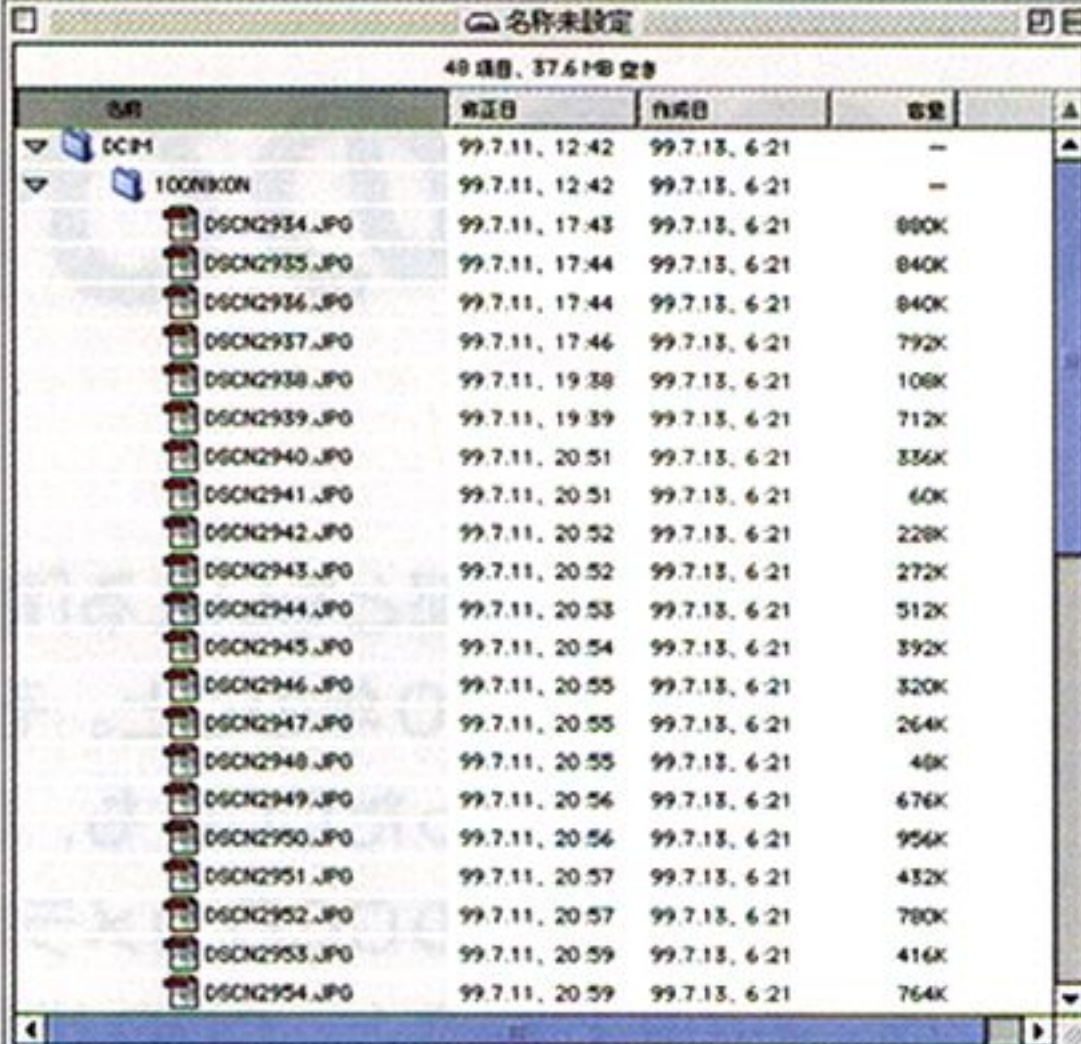
でもバッファがいっぱいになった途端にカメラが処理を受け付けなくなってしばらく待たねばな

らない機種もあり，まだまだ完璧とはいえない。
富士写真フイルムはそういうバッファをいまのところ持たずに頑張っている。だから数字上の撮影間隔は長いし連写には向かないが，1枚につき5秒くらいなら許容限度内だろう。
サンヨーのDSC-SX150は，画像処理からJPEG圧縮，記録メディアへの書き出しまでを1チップに集積し，なおかつμiTRONを使うことでマルチタスク化して高速化を図っている。つまりバッファにため込むということをせず，正味1〜2秒で1枚分の処理が完了するのだ。偉いものである。

・スマートメディアとCFカード

現在ほとんどのデジカメは記録メディアとしてスマートメディアかコンパクトフラッシュカードを採用している。FDとかメモリースティックを採用しているのもあるけど，それは割愛ってことで。
スマートメディアは東芝が開発した薄型フラッシュメモリカードで，とにかく薄くて軽くて風で飛んでいきそうである。富士写真フイルムが「デジタル版のフィルムとしていけそうだ」と最初に採用し，そのあとオリンパスも採用。日本の2大デジカメメーカーが採用したため，普及枚数は多い。
フラッシュメモリカードは，メモリチップとそれを駆動するためのコントローラが必要になる。
スマートメディアはメディア側にチップだけを載せ，コントローラは本体(あるいはPCカードアダプタなど)に持たせたため薄型化でき，構造がシンプルになるため製造コストも安く抑えられるというメリットが生まれた。だが，メモリチップの大容量化によって「古いコントローラだと動作しない」という事態が発生し，しかもそのコントローラは「カメラ側」が持っているため，古い機種ではいまの16MB以上のメディアが使えないとか，PCカードアダプタも古いものは8MBのメディアまでしか対応しないというドタバタにつながっている。
スマートメディアの容量はいまのところ32MBのメディアが最大だが，今年中に64MBまでいきそうな気配である。
コンパクトフラッシュカードはCFカードと略されることが多い。米SanDiskが開発したもので，要するにフルサイズのTypeIIフラッシュメモリカードを半分以下のサイズに小型化しただけである。よって，ちょっと分厚くて，コントローラまで内蔵している。その分スマートメディアに比べて製造コストがかかるはずだが，世の常として，そういうのはあまり当てにならず，けっこう安く出回っている。いまのところ96MBまで出ているのかな。安くなっているとはいえ，1MBあたりの単価ではまだスマートメディアのほうがちょっと安い。
これは容量がでかい・互換性が高いというメリットがあるわけだが，実際にはメモリの種類やら4倍速，8倍速やら微妙な違いが生じていて，カメラによってはうまく動作しないものもある(CoolPix950などでトラブルが発生している)。安全を期するなら米SanDisk製のものにするのがいいだろう。
どちらもフラッシュメモリなので書き込み速度はさほど速くない。CFカードは倍速とか4倍速といっているだけあってけっこうモノによって速さに違いがあるようだが，その恩恵にあずかれるのは十分高速なPCカードドライブを使ってPCに画像を転送するときくらいだろう。

・DCFとExif

以前はカメラによってメーカーによってまちまちだったが，いまはほとんどすべてのデジカメがDOSフォーマットの記録メディアにJPEGフォーマットで画像を記録する。拡張子もちゃんとjpgである。よって，PCカードリーダーを使えばWindows上にはリムーバブルメディアとしてマウントできるし，MacOSでもDOSフォーマットのメディアとしてマウントできる。
だが，いくつかのメーカーや日本電子工業振興協会が動き出し，もっと細かいところまでルールを決めましょうってことになった。やがて，富士写真フイルムが中心となり，日本電子工業振興協会(JEIDA)の標準となった「Exif」(イグジフ)と，キヤノンが中心となった「ciff」(シフ)の2種類の基準ができ，今年になってそれらが「DCF」という名前で統一された。
DCFは「Design rule for Camera File system」の略である。よくわからん略し方だがまあいいや。
DCFでは「ファイル管理」と「ファイルに持たせる属性(タグ)」の両方を規定している。ファイル管理のほうはフォルダ名やファイル名のルール，ディレクトリ構造のルールが定められている。ルートディレクトリには「DCIM」というフォルダを持ち，そのなかには「3ケタの数字＋5文字の英文字」のフォルダを持ち，その中に画像ファイルを入れなさいって感じだ。
画像ファイルに持たせる属性はExif 2.0をベースとして，160×120ドットでJPEGのサムネイルデータを持たせようとか，撮影日時とかをこういう形式で入れようとか，そういうファイル内部の構造が規定されているわけだ。オプションで絞り，シャッタースピードなど撮影情報も入れられるようになっており，ニコン，オリンパス，富士写真フイルムなど多くのメーカーはいろんな情報を書いてくれている(逆に，ソニーのCybershotはほとんどオプションのタグを使ってくれてない)。
DCFのおかげで，画像ビュアを作るときの基準ができたし，メディアを見ただけでそれがデジカメの画像が入っていてしかもどのカメラで撮ったのかなどがわかるようになる。今後いろいろと便利なソフトが育っていくだろう。

・JPEGとTIFFとCCD-raw

ここではJPEGで圧縮して記録する方法について述べたが，実はJPEGといっても圧縮率を変えることでファイルサイズを変更できるし(この圧縮率の違いをデジカメでは画質の違いとしている)，業務用途に向けて非圧縮のフォーマットも用意している。
デジカメが採用している非圧縮フォーマットの代表がTIFFとCCD-rawだ。CCD-rawというのはCCDからきた信号の形式のまま保存するもので，RGBの素直な並びにはなってない。前のほうで述べたように，デジカメではCCDの各画素にそれぞれカラーフィルタをつけていて，足りないところはあとで補間することで画像を作っている。CCD-rawは補間前の信号ってことだ。よって，ファイルサイズも非圧縮のベタな画像(非圧縮TIFFなど)に比べて小さいのである。ただし専用のソフトがないとそれをRGBの画像に展開できない。
TIFFはデジカメ内で画像を作ったあと，非圧縮のTIFFフォーマットで保存するもの。この場合はファイルサイズはもっとも大きくなるが，標準フォーマットのメリットがある。
まあいずれにせよ，JPEGに比べて画像ファイルの書き込みに時間がかかるしファイルサイズも著しくでかくなるのでよほど圧縮によるノイズを嫌うケース以外では使う必要はないだろう。

図11　DCFに準じたファイル管理の例：Nikon COOLPIX950

以上，長くなったけれども，レンズから記録メディアまでカメラとデジカメの基本を流してみた。銀塩もデジカメの基本原理は一緒なのだ。ただ，フィルムという化学な世界と，CCDという電気な世界が違うだけなのである。よって，カメラについて知ることができれば，デジカメと銀塩カメラを無謀な観点で比較することもなくなるし，両者を適材適所で使い分けることが可能になるし，単なるデジカメユーザーにとっても撮影時の指針になるはずだ。また，基本動作がわかっていれば，撮影した写真をレタッチするのも容易になる。
デジカメと銀塩は似ているようで異なった長所短所を持っている。その辺を理解すれば百戦危うからずである。

参考文献：「EF Lens Work II」(キヤノン)，「写真用語辞典」(日本カメラ社)

CCDのひみつ

西川善司 Nishikawa Zenji

デジカメの心臓部ともいえる「電子の目」，それがCCDだ。「CCD」というものが米国のベル研究所で発明されたのが1970年のこと。そして民生品の製品用デバイスに降りてきたのは1985年のことだ。当時は高級デバイスだったCCDも，いまではノートパソコンにおまけ機能的に内蔵されたり，1万円以下のビデオチャット向けのCCDカメラなんかにも搭載されたりしている。非常に身近になったCCDだが，ここでは，その動作概念と，CCDが実際にどうデジカメにインプリメントされているかを見ていくことにする。

撮像素子の歴史～撮像管とは

そもそも，映像を電気信号に変える「撮像素子」にはどういうものがあるのだろうか。CCDが発明される前からテレビ放送はテレビカメラでとらえた映像をみんなのところに電波で飛ばしていたわけだし，その「映像を電気信号に変える仕組み」自体はずいぶん前からあったことになる。

もっともその歴史が古くメジャーな存在なのが「撮像管（Image Pickup Tube）」と呼ばれる撮像デバイスだ。これはドイツの物理学者ヘルツが発見した「光電効果（photoelectric effect）」という現象が基本原理になっている[*1]。

光電効果とは半導体や金属に光（放射）を当てると自由電子を生ずる現象だ。具体的には，光子（Photon）エネルギーを半導体や金属中の自由電子が受け取り，これが飛び出す。この電子を光電子（Photo Electron）といい，これをキャッチすることができれば電流を流すことができる。この電流の強弱が入射光の強さに比例するため，これはつまり，光量を感じ取るセンサとして利用できることになる。

この現象を真空管で実現した，もっともシンプルなデバイスが光電管だ（図1）。

さらにこの光電管の概念を用いて撮像デバイスを構成したのが「撮像管」ということになる。

撮像管の仕組みはこうだ。

光電面に三硫化アンチモン（Sb2S3）や一酸化鉛（PbO）といった材質の光電膜を構成し，ここにレンズで映像を絞り投影させる。ここを電子ビームで走査（スキャン）すると，その部分の光の明暗に応じた光電子が飛び出してくるので，これを集束，あとはこの電気信号を再編成すれば，これは二次元的な映像として得られることになる（図2）。

こうした撮像管にはビジコン[*2]，サチコン[*3]といったものがあり，CCD全盛の現在でも使われている。NHKのHARP撮像管（High-gain Avalanche Rushing amorphous Photoconductor）はハイビジョン放送用ビデオカメラに採用されている[*4]。

CCDの登場

撮像管の欠点としては「耐久性が低いこと」，「過度の光を入射させてしまうと光電膜が焼きつきを起こしてしまい，それ以後正しい映像を出力できなくなってしまう」点などが挙げられる。

そこで開発されたのがCCDというデバイスだ。今度はCCDの動作概念を見ていこう。

CCDってどういう意味？

ところで，CCDとはなんの略なのだろうか。

よく，マルチメディア用語辞典や技術系雑誌や工業系新聞などで「CCD」という言葉をサーチすると，

CCD＝固体撮像素子

という表現で見かけるが，これではなんのことかわからない。「固体」はSolid Stateだし「撮像素子」とはImage Sensor Deviceだから，SSISDとなりCCDの略記と一致しない。

もうひとつよく用いられる表現として，

CCD＝電荷結合素子

というのがあるが，こちらのほうがオリジナルの意味だ。「CCD」とはCharge Coupled Deviceの頭文字を取ったものなので「（Electric）Charge＝電荷」「Coupled＝結合された」で意味として合致する。

しかし，「君のデジカメの電荷結合素子はどん

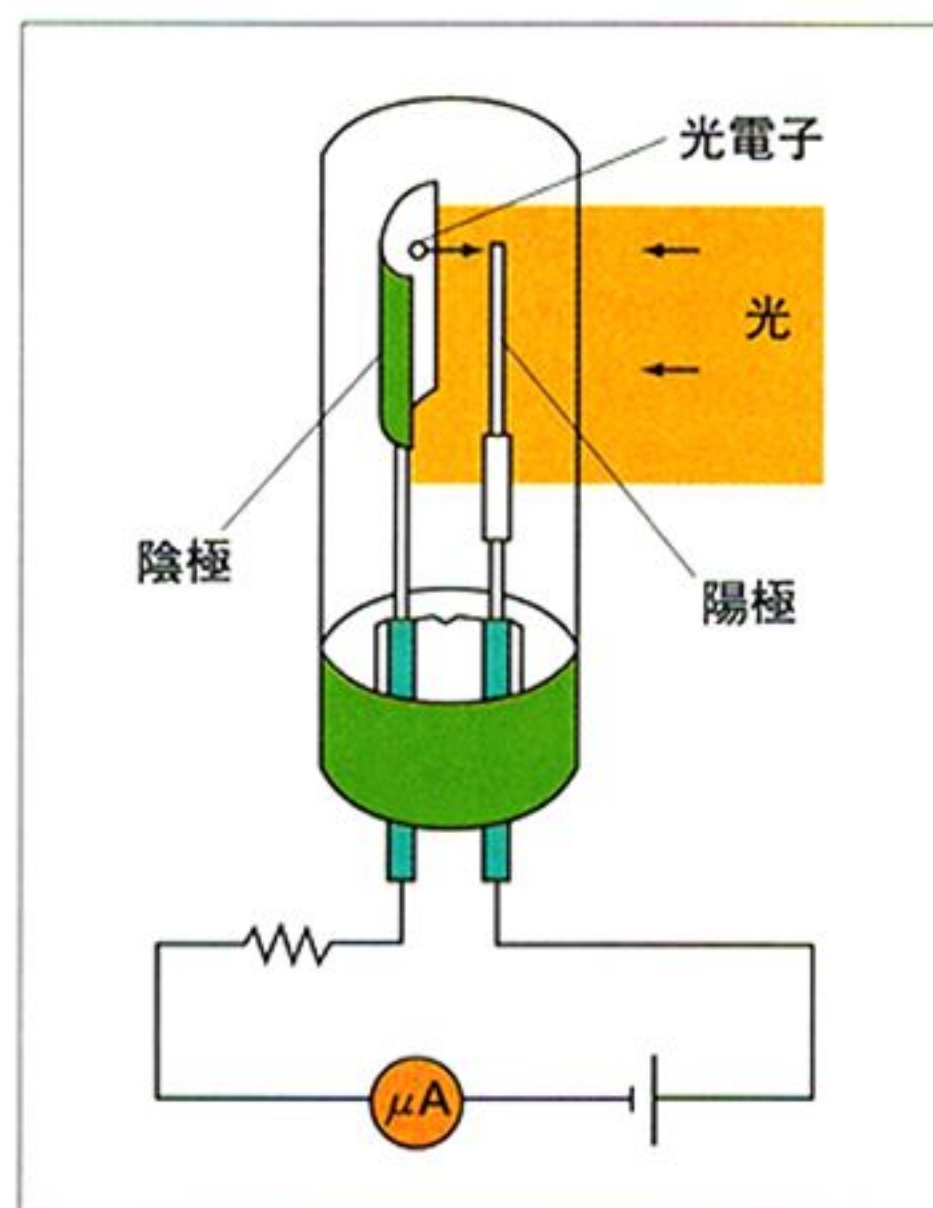



図1　光電管の構成

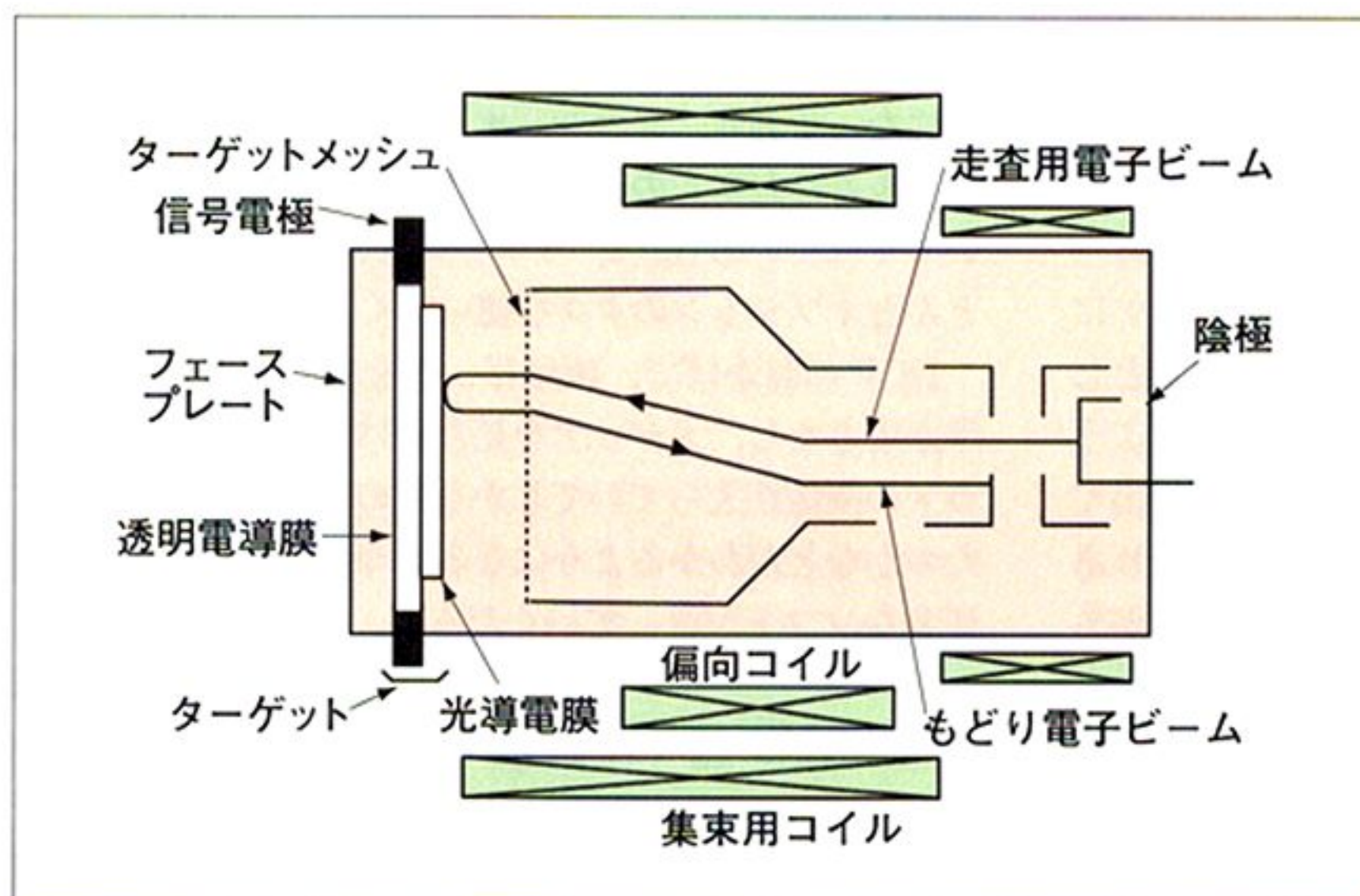



図2　撮像管のようす

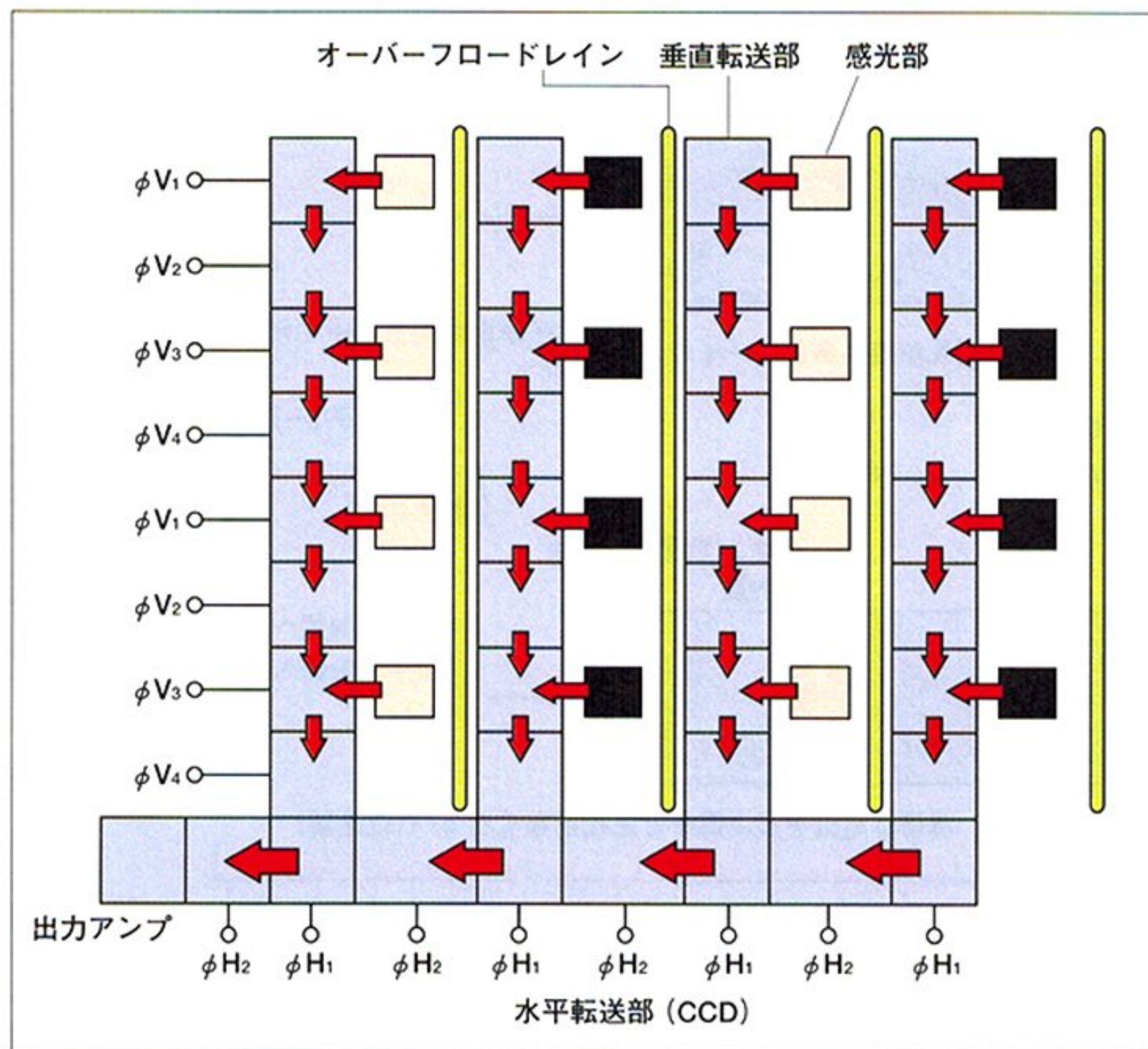



図3-a　CCD内の電荷の流れ

図3-b　CCDの構造

な感じ?」と友達に聞いても口をあんぐり開けたまま眠りこまれてしまう可能性があるほど，直感的にはわかりにくい。

実は，CCDという言葉自体には「撮像（イメージセンサ）」の意味はなく，素子自体の電荷の伝達方式を示す名称なのだ。具体的には「出力として得られた電荷が繋がった状態で取り出されます」という意味がCCDという言葉には込められているのである。

では「固体撮像素子」とは，どういういきさつで生まれた言葉なのだろうか。これは前出の撮像管に対する言葉として生まれたもののようだ。撮像管はビデオカメラに用いられるほど小さくなったとはいえ，その構造は「立体的な小実験場」のように高度で複雑だ。これと比べればCCDは，たった1個の半導体デバイス（1チップ）で実現できるため「固体」とみなすことができる。

(*1) 光電効果をプランクの量子説を取り込んで光を粒子として説明したのがかの有名なアインシュタインだ。
(*2) 1950年ごろアメリカのRCAで開発された撮像管。美人コンテストのことではない。
(*3) 1974年頃にNHKと日立製作所が開発した撮像管。サッチーコンプレックスのことではない。
(*4) HARP管の最新型新super-HARP管についてはNHKのWeb（http://www.strl.nhk.or.jp/publica/dayori/dayori97.04/kaisetsu1-1-j.html）を参照してほしい。

CCDの受光部の構造

それでは，今度はCCDがどうやって映像を電気信号に変えていくかを見ていくことにしよう。やはりCCDも基本原理としては光電効果を利用している。

図3はCCDの概念図だ。まずは受光部について見ていくことにしよう（図3-b）。

光は図の受光部から取り入れられることになるわけだが，これ以外の部分は薄いアルミニウムなどの材質でシールドされている。受光部以外から光が入射すると，これがノイズの原因になるからだ。

受光部の一番上の表面は窒化シリコンなどの材質でコーティングがされている。いわゆる保護膜のようなものだ[*5]。

そしてこの保護膜の下に，撮像管の光電膜に相当する，半導体を持ってくる。この材質にはシリコン（Si）が使われる。実際にはポリシリコン（多結晶のSi）で，この中にアンチモン（Sb）や燐（P），砒素（As）といった不純物を混ぜている。これが半導体を形成する重要な仕組みで，この不純物のことをドナー原子という。ドナーは提供者（Donor）のことでリーサルウェポン・シリーズの映画監督とは無関係だ。どうしてドナー（提供者）なのかはあとでわかる。

さて，シリコンの最外殻電子数は4，ドナー原子の最外殻電子数は5だ。お互いが共有結合した場合[*6]，ドナー原子側の電子がひとつ余る計算になる。この電子は自由電子になりやすい。ちょうどこの部分は電気的にマイナス（ネガティブ：Negative）なのでN型半導体になっているわけなのだ（図4）。

そして，このN型シリコンの下側には二酸化シリコンの膜が位置している。これはいわゆる絶縁体だ。電気は通さない代物だ。ただし，電界の影響には無力なのはいうまでもない。

ちなみに，窒化シリコン，N型シリコン，二酸化シリコンとも透明の材質で光をよく通す。

この二酸化シリコンによる絶縁膜の下には，またまた，ポリシリコンがある。今度も不純物が入れられており，今度のは最外殻電子の数がシリコンよりも少ない3の原子が入れられている。具体的には硼素（B），インジウム（In），ガリウム（Ga）などを用いることが多い。

こうした不純物がシリコンと共有結合した際には[*7]，電子が1個足りないため，正孔（Hole）を作ることになる。正孔とは電子を「受け取る」

図4

図5

(Acceptする)意味あいがあるので，この場合の不純物原子をアクセプタ原子という。ちょうど，この部分は電気的にはプラス(ポジティブ：Positive)なのでP型半導体になっていることになる(図5)。

(*5) ポリシリコンは波長の短い青色光を通しにくい特性があり，この上の窒化シリコン層はこの特性を弱める意味あいもある。
(*6) たとえば燐(P)の場合ならばM殻での共有結合となる。
(*7) たとえば硼素(B)の場合ならばM殻での共有結合となる。

CCDの受光部の動作概念

P型半導体とN型半導体が絶縁体を境に並んでいるので，これはちょうど「PN接合」と呼ぶフォーメーションになる。

このとき，P型N型の接合部分付近ではP型側の正孔とN型側の自由電子が互いに混じりあって拡散結合をしていく(図6)。

このとき，P型のほうのアクセプタ原子は正孔が出ていった(電子をもらった)ために(－)イオンに，一方N型のほうのドナー原子は電子が飛び出したことから(＋)イオンに変貌してしまう。

つまり，これはP型にマイナスの電界，N型にプラスの電界が作られることになるわけで，しばらくするとP型側の正孔はN型側のプラスの電界に弾かれ，N型側の電子はP型側のマイナスの電界に弾かれ，PN接合部付近には電気を運ぶ電子と正孔がなくなる現象を引き起こす。

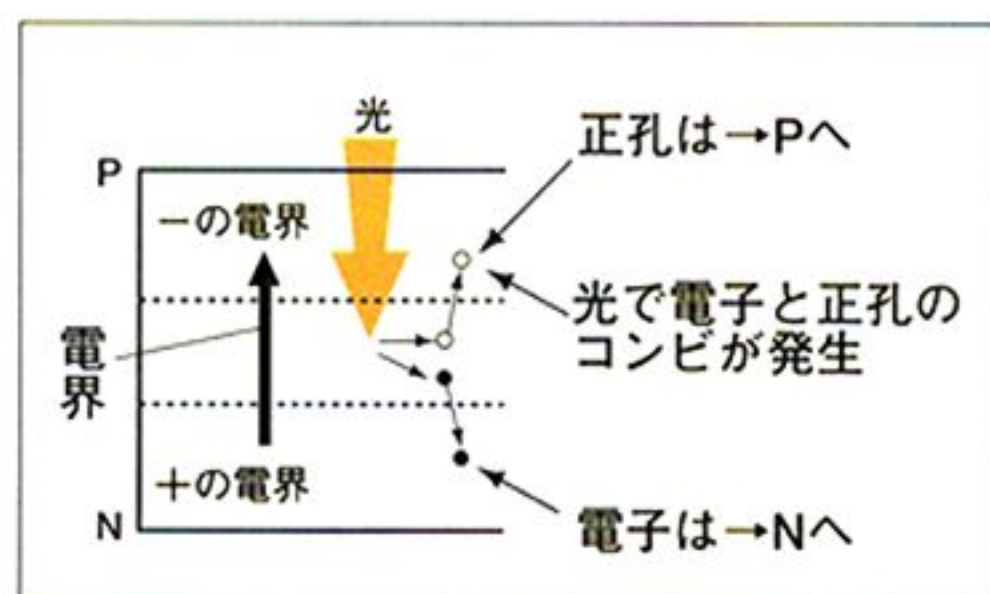
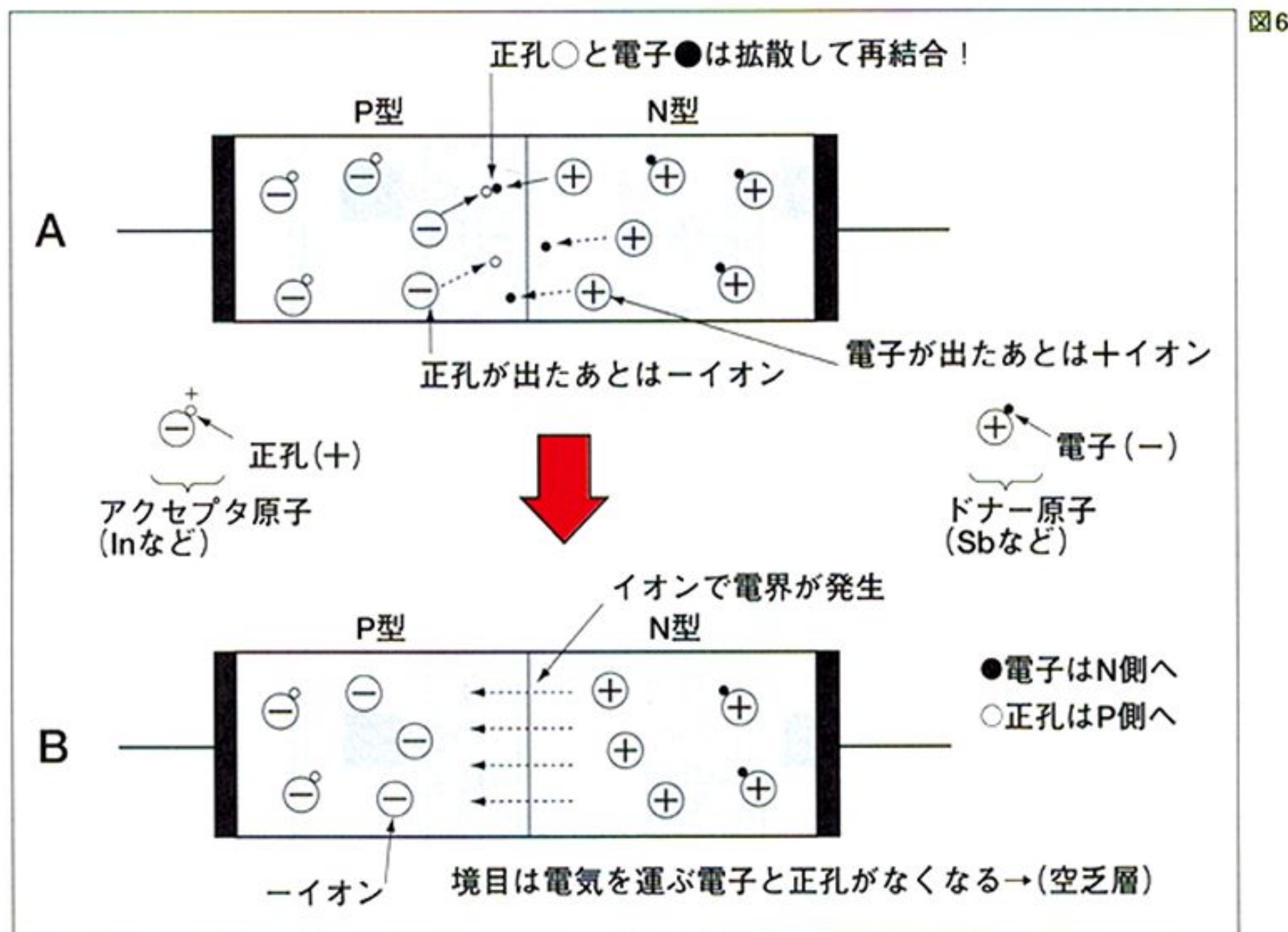
この領域のことを特に「空乏層(くうぼうそう)」と呼ぶ。

さて，この状態で光が，受光部から入ってきたとしよう。

光は窒化シリコンの保護膜を抜け，ポリシリコンのN型半導体を抜け，二酸化シリコンの絶縁膜(PN接合部)にまで到達する。

光の粒子，光子はエネルギーを持っており，これが空乏層付近の原子中の電子と衝突すると電子が飛び出し，正孔が発生する。そう，これが撮像管のところでも触れた「光電効果」だ。

飛び出した電子(－)は電界の作用でN型のほうへ，正孔(＋)のほうは同じく電界の作用でP型のほうへと移動し，誘導電流が流れる。

光が多ければ，すなわち明るければ多くの電流が流れることになり，暗ければその量は小さくなる。こうして「光が当たるとその量に応じて電流が発生する」という機能を実現しているのだ(図7)。

ただ，せっかく飛び出した電子も，電界の力が弱いと，元の正孔に戻って消えてしまうことがある。つまり，これは「受光素子として感度が悪い」ということになる。

これではまずいので，「電界を強める」→「すなわち空乏層の厚みを広げる」という意味あいからP型のほうにマイナス，N型のほうにプラスの電圧をかける工夫を行っている(図8)。

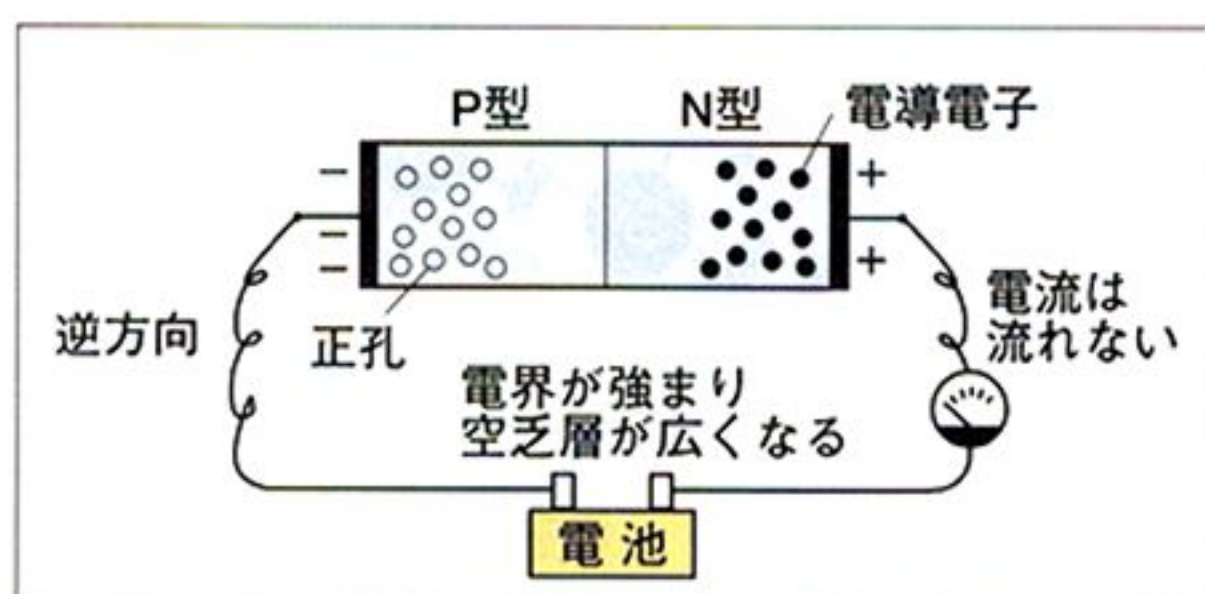
受光部の電子を取り出す

続いて，この電子を転送部の電極で転送するプロセスに移る。

前出の図3-aをもう一度見てほしい。

光の量に比例した電子の量をいかに正しく出力できるかが，この部分の「腕の見せどころ」になるわけだ。

受光部で発生した電子はここにかけられた電圧のもとにたまっている。この電子をまずは垂直転送部のほうへ移す作業を行うのだ。

垂直転送部は電線ではなく，P型シリコン単結晶基板にシリコン酸化膜(二酸化シリコン)を載せ，アルミ蒸着膜の電極が少しずつ離れた状態で並んでいる。実はこの部分の構造こそが「CCD」であり，すなわち「電荷結合素子」であるのだ。だから，この転送部を「垂直転送CCD」と呼んだりもする。

さて，受光部の電子をここにまず移すことを行うわけだが，これは受光部の電極電圧よりも一瞬，大きい電圧をここに作ってやることで行う。

いま，このアルミ電極にプラスの電圧をかけたとすると，酸化膜を挟んでその下のP型シリコン基板の上層の正孔はこのプラスの電圧に反発して押しのけられ，ここに空乏層が発生する。この電極直下の領域は電子が入り込みやすい部分であり，その「入り込みやすさ」は電圧の大きさに比例する。この領域は電子の入り込みやすさを井戸の深さにたとえて，特に「電位の井戸(Potential Well)」と呼ばれている。

受光部の電極電圧がたとえば3Vだったとすると，この受光部の電子をこの垂直転送CCD側に移し込むには，5Vの電圧をかけたりすればいいのだ。

電子の転送

さあ，ついに受光部の電子が垂直転送CCDに移動した。今度は電子を下流の水平転送CCDのほうへ移動していくことを考えなければならない。

この作業もやることは同じ。電位の井戸の深さを巧みに調整してバケツリレーのようなステップで電子を移動させていく。

図9を見てほしい。この図は三相電極を用いたCCDの概念図だ。[*8]

それぞれの電極が独立した電圧を供給できる電源に接続されている。

図中の(i)において，A，A'，A"が受光部と接続されており，ここに，受光部の電極電圧よりも高い5Vの電圧をかけたとしよう。これはずばり前段で述べた部分に相当するところ。受光部の電子がA，A'，A"の電極に移動する。

続いて次のフェーズ(ii)では隣の電極であるB，

図6

図7

図8

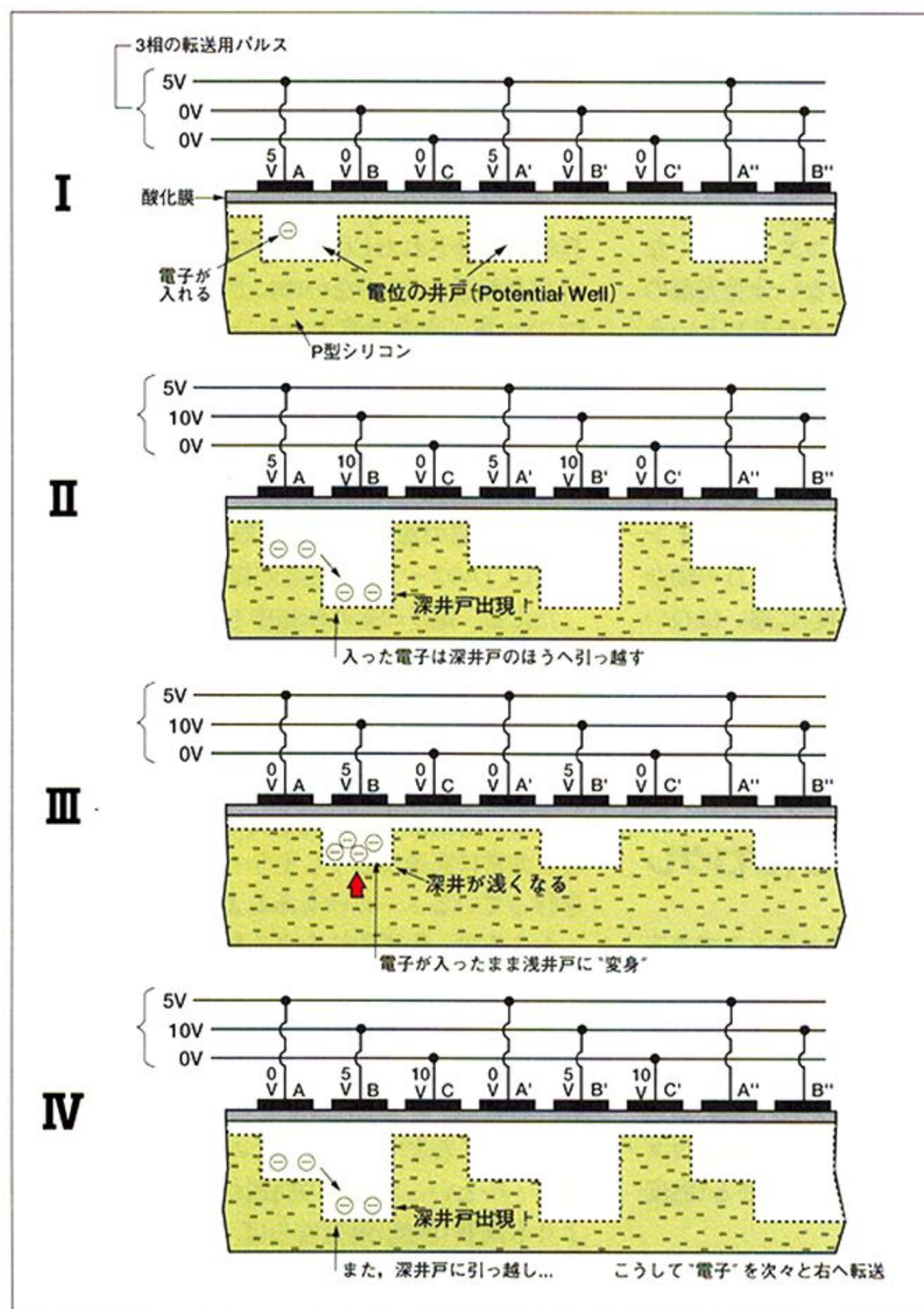



図9

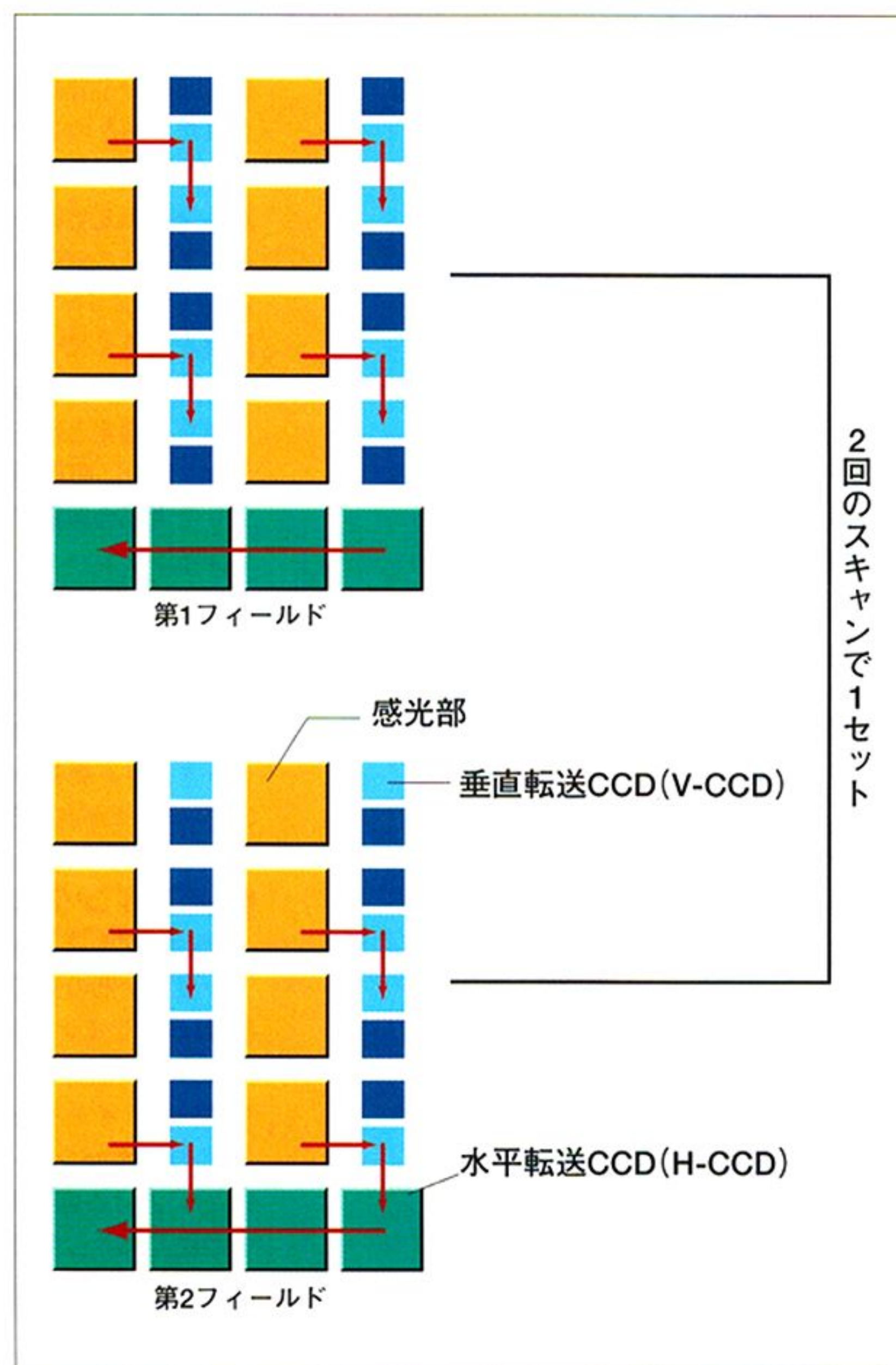



図10

B', B"をより高い電圧，ここでは+10Vに設定すると，電位の井戸がより深くなるため，電子はそちらへ移動する。

10Vだった電極B, B', B"の電圧を今度は5Vまで下げてやり（iii），電極C, C', C"の電圧をこれよりも高い10VにするとB, B', B"のところにあった電子は今度は電極C, C', C"の元へ移動する（iv）。

あとはこれの繰り返しだ。

垂直転送CCDから水平転送CCDへの電子の移動も，水平転送CCD内での電子の転送もあとは同じ原理で動作する。

水平転送CCDの出口には出力アンプがあり，得られた電子の数を電気信号に増幅する工程を行うことになる。

水平転送CCDの出口では流れ作業的に，ある決まったシーケンスで電子が出てくるので，いま獲得した電子がどの受光部の出力かというのは容易に把握できる。「面」としてとらえたデータを水平転送CCD，垂直転送CCDを利用して，一次元的なひも状のデータに再編成している……と考えればわかりやすい。

（*8）二相電極のものや仮想電極を用いた単相電極のCCDもある。

インターライン転送型CCD(ILT-CCD)

ここまで解説してきたCCDは「インターライン転送型CCD」(ILT-CCD)と呼ばれるものだ。

「どういうタイミングで垂直転送CCDに受光部の電子を移すか」というのが，そのままカメラのシャッターの動作に相当するため，デジカメを構成するときに機械的なシャッターが不要になる。ILT-CCDのこの機能を「電子シャッター」と呼んだりする。

ところで，画素情報のスキャン手法の違いからILT-CCDにはインタレースCCD(*9)とノンインタレースCCD（プログレッシブCCD）の2通りに分かれる。

（*9）ときどき，インターライン転送型CCD（ILT-CCD）とインタレースCCDという言葉が似ているため，同義と誤解されやすいが，ILT-CCDが全体集合の名称である点に注意したい。インタレースCCDもノンインタレースCCDもILT-CCDなのだ。

●インタレースCCD

インタレースCCDは画素の電荷を1ラインおきに読み出すCCDだ（図10）。画像表示方式のインタレースと意味あい的には同じだ。

インタレースCCDはもともとビデオカメラ向けに作られたものだ。NTSC規格のテレビ（ビデオ）の場合，各フレームは1/60秒おきに描かれているものの，その瞬間瞬間は，奇数ラインのスキャンしか行われていない奇数フィールド，偶数ラインしかスキャンされていない偶数フィールドが表示されている。インタレースCCDのスキャンタイミングを1/60秒と設定すれば，その瞬間瞬間に得られたフレームを奇数フィールド，偶数フィールドとして記録していけばちょうどいい。

ただし，デジタルスチル（スチル＝STILL＝静止画）カメラの場合は少々状況が異なってくる。

インタレースCCDでは1回の走査で得られる画像は画像全体の情報量の半分でしかないため，2回の走査が必要になるわけで，もし被写体の動きが非常に速い場合，図11上のような映像になってしまうことがあるのだ。

こうした現象を取り沙汰され「インタレースCCDはデジカメ向きじゃないよねぇ」ということをいわれるようになってしまったが，実は一概にそうともいえない。

まず，被写体の動きが非常に速い場合に起こる「1ラインごとのずれ」だが，これはCCDの電子シャッター機能に頼るからこういうことになるだけ

インタレーススキャン

動き

プログレッシブスキャン

動き

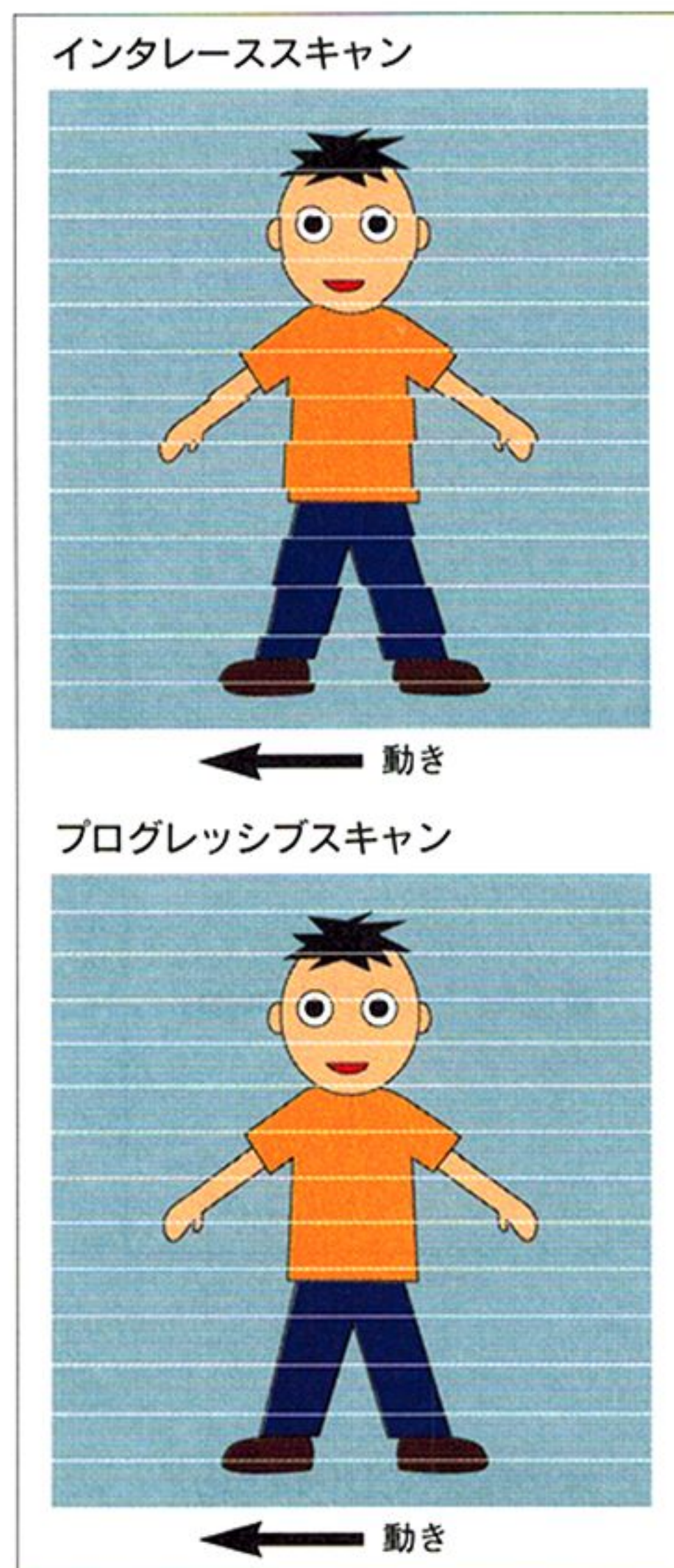



図11

で、CCDに光を入れたあと完全に遮光するような(普通のカメラにあるような)機械式シャッターを組み合わせることで簡単に解決できる。

CCDの受光部に入射した光によって生まれた電子は、少しの間は保持されるので、光を入れたあとすぐに遮光してしまえば最初の画面半分の画像情報を読み出している間も、受光しないですむ。最初の半分を読み出し終わったら、その保持された、あとの半分を読み出せばいい。

インタレースCCDのよい点は各受光部付近を走る垂直転送CCDの電極の個数を減らせるところにある。これはつまり、「製造コストが安くなる」というメリットと、「電極が少ない分受光部の面積を大きくできる(=感度がよくなる)」というところに繋がるからだ。

数百万画素という製品が出てきている最近のデジカメだが、画素が高密度化するということはつまり受光部の面積が小さくなるということでもある。インタレースCCDにしろ、後述のノンインタレースCCDにしろ、同じ画素数のCCDの場合ならば受光部の数(すなわち解像度)に違いはないので、感度がよいほうがいいに決まっている。デジカメにおけるCCDを考えた場合、機械式のシャッターさえ組み合わせてしまえばインタレースCCDのほうが有利ということもあるほどなのだ。

●ノンインタレースCCD(プログレッシブCCD)

一度に全部の受光部から入力をスキャンできるCCDがノンインタレースCCDだ(図12)。

受光部の電子を一斉に垂直転送CCDに転送できるため、インタレースCCDのように2回に分けてのスキャンは不要だ。その分、垂直転送CCDにおける電極の数が増えるため、「製造コストが高くなる」「受光部の面積が小さくなる(感度が悪くなる)」という特徴がある。

フレーム転送型CCD(FT-CCD)

ILT-CCDに対して、もう一つ別のタイプのCCDがある(図13)。

見かけ上は垂直転送CCDを持たず、各ラインの出力を水平転送CCDにまとめて送り込み、1ラインごとの出力を得られるCCD、「フレーム転送型CCD」(FT-CCD)と呼ばれるものだ。

実はILT-CCDよりも古典的な手法で1970年当初に開発されたCCDもこの方式だった。

「垂直転送CCDがないのにどうやって水平転送CCDへ電子を受け渡すのか」という疑問がわくわけだが、これは受光部が垂直転送CCDを兼ねており、受光したあと出力として得られた電子をここをCCDとして使って転送していくのである[*10]。

垂直転送CCD用の電極が一切いらないことから、受光部の面積を大きく取れるため、なんとILT-CCDの2～3倍の感度が得られるのが特徴だ。

ただし、電子の垂直転送中も受光部は働き続けていることから、転送中にも受光部の出力を受け取ってしまう弱点もある。これはインタレースCCDと同じく、機械式のシャッターで光を入射させたあと遮光してしまえば解決できる問題だ。

あるいは転送中の受光を無視できるほど転送速度を非常に高速にしてしまう……という解決方法を行っているメーカーもあるようだ。

ところで、転送手法をよく見て考えるとわかるが、FT-CCDではILT-CCDのように出力をパイプライン的に得られないため、出力の転送速度がそれほど速くないというデメリットもある。ただし、これは、FT-CCDでとらえた映像を液晶ディスプレイなどでリアルタイム表示するのに向かない……といった程度の問題だ。リアルタイム性よりも画質(感度)優先ならば、ILT-CCDよりもFT-CCDを選択するほうが有利なときもある。

(*6) 受光部で得た情報をその画素分用意した蓄積分に一気に転送してから、水平転送CCDで出力を取り出すような方式もある。

MOS/CMOS型イメージセンサ

CCDという撮像素子は受光部で得た電子をCCDと呼ばれる電極で転送していた。これとは別の方法で電子を転送する撮像素子も早くから開発されている。代表的なのが1981年に実際に製品化されたMOS型イメージセンサと呼ばれるものだ。

ところで「MOSっていうのは金属(METAL)、酸化物(OXIDE)、半導体(SEMICONDUCTOR)のことだから、それをいったらCCDだってアルミニウム(M)、二酸化シリコン(O)、P型シリコン(S)という3層構造のデバイスで電子を転送していたわけだからMOSでしょ?」という意見も出てくるだろう。

ところが撮像素子のタイプ名として「×××型イメージセンサ」と呼ぶ場合にはその電子の転送方式に着目して呼ぶらしいので、CCDはあくまでCCDと呼ぶようだ。

それでは今度はこのMOS/CMOS型イメージセンサ方式についての仕組みを見ていくことにしよう。

図12

水平転送CCD

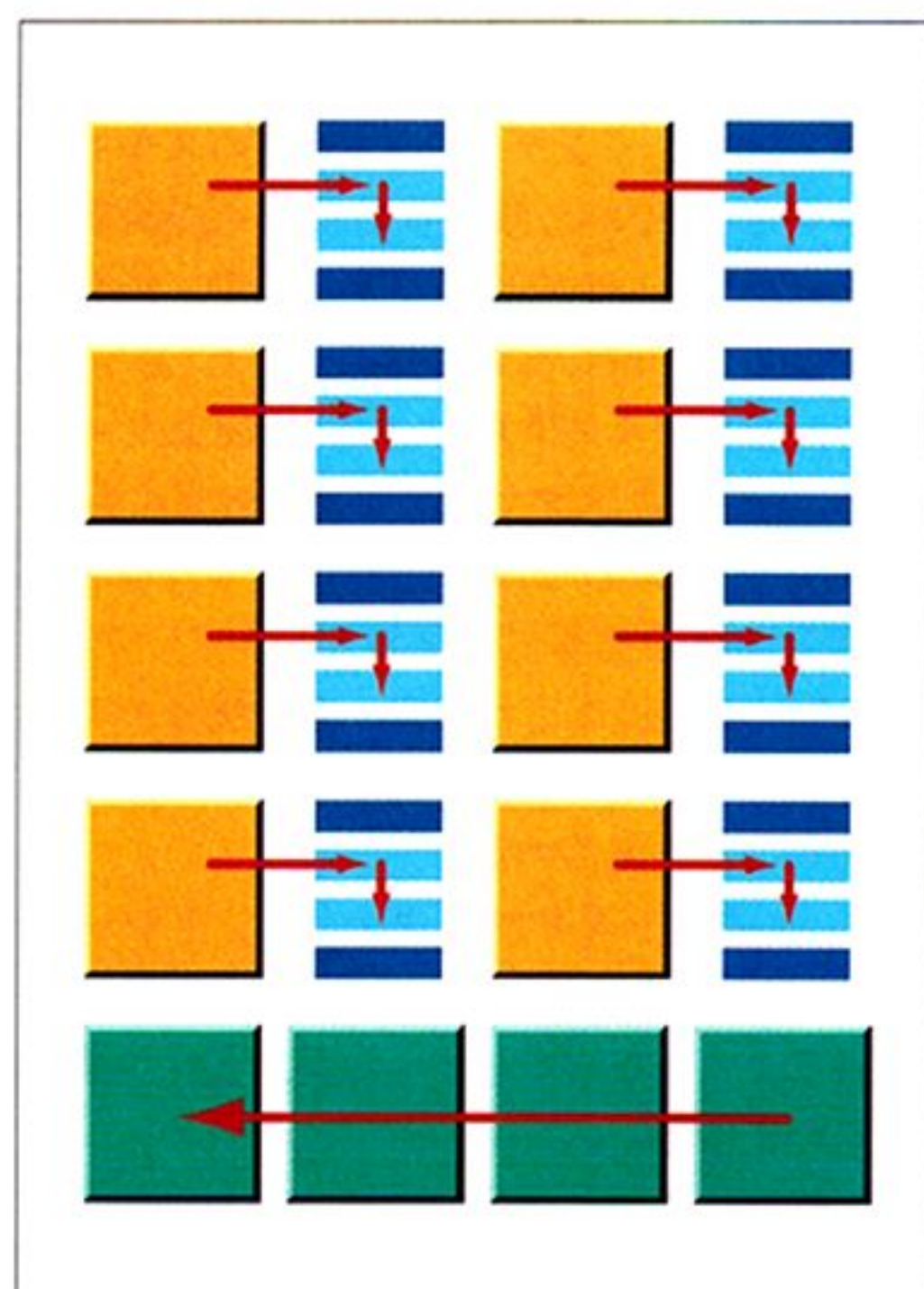
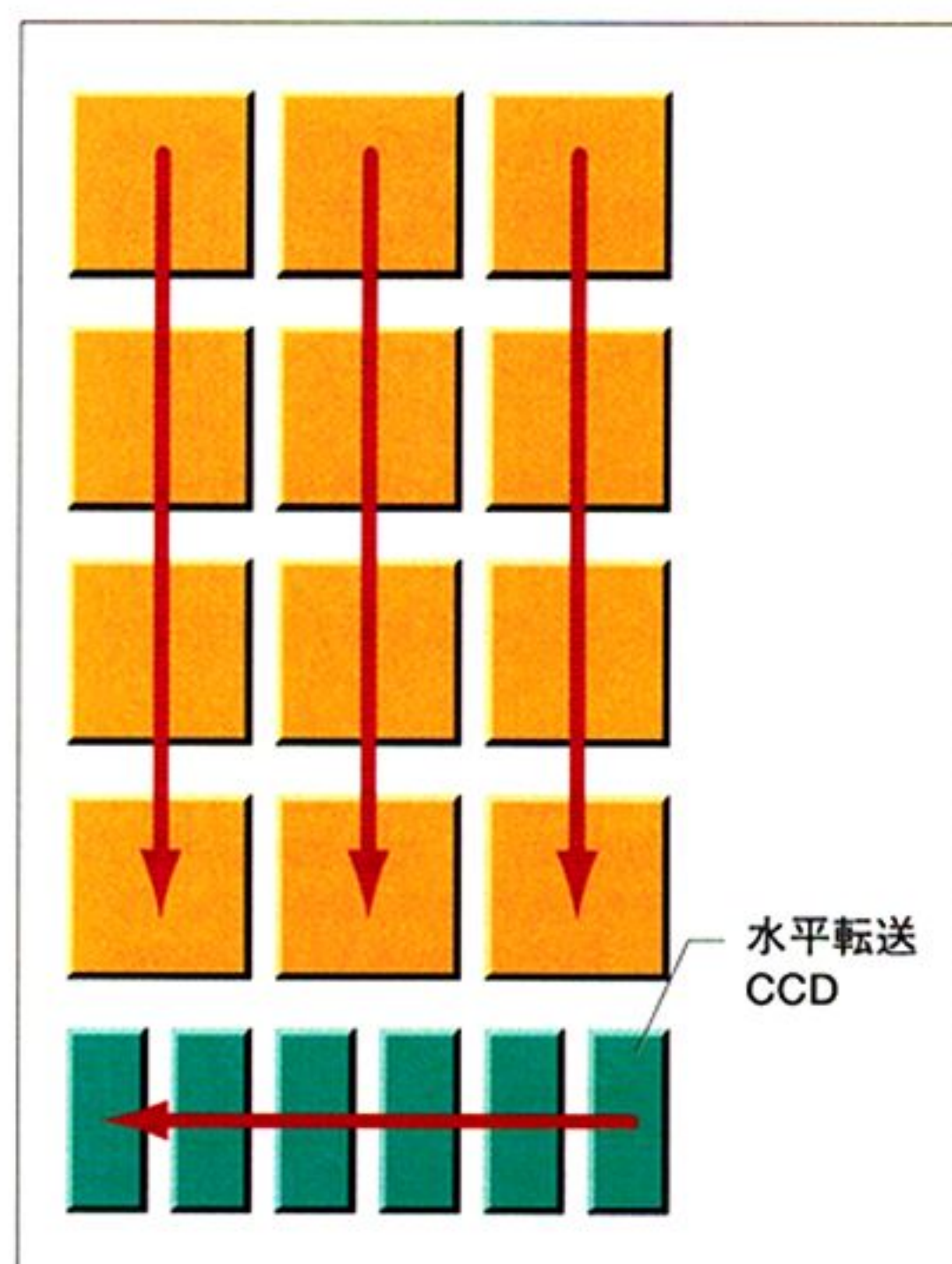



図13

MOS/CMOS型イメージセンサの構造

まず，受光部だが，その仕組みと動作はCCDと同じと考えてもらっていい。

受光部にはそれぞれの専属のMOS-FET(電解効果トランジスタ)が接続されていて，具体的には図14-aのような構造をしており，これを等価回路で表すと図14-bのようになる。この受光部に接続されたMOS-FETは後述する垂直スイッチング用のMOS-FETとして機能する。

そして，この垂直スイッチング用のMOS-FETは受光部とは別のところに設置された水平スイッチング用のMOS-FETに接続されている。

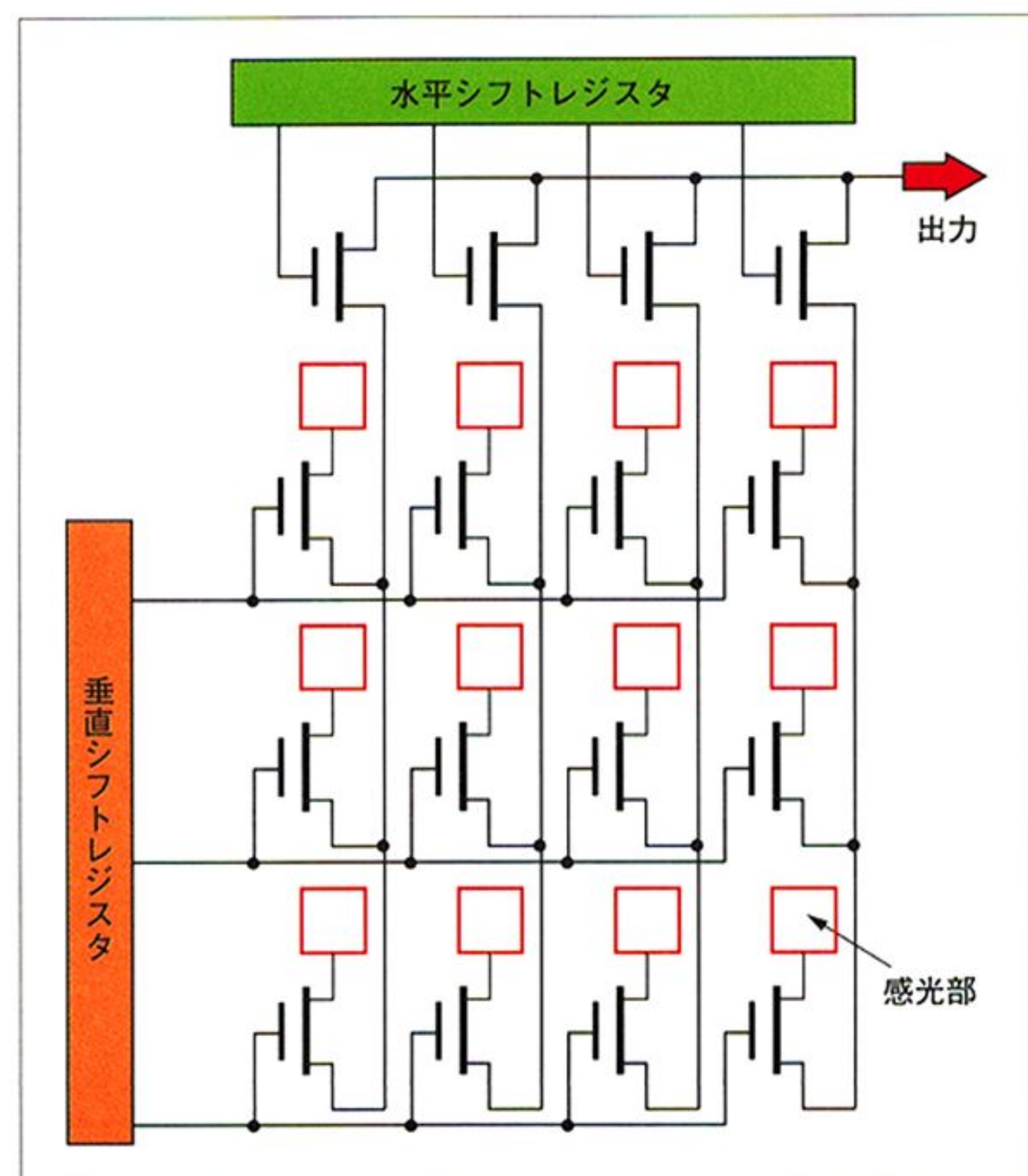
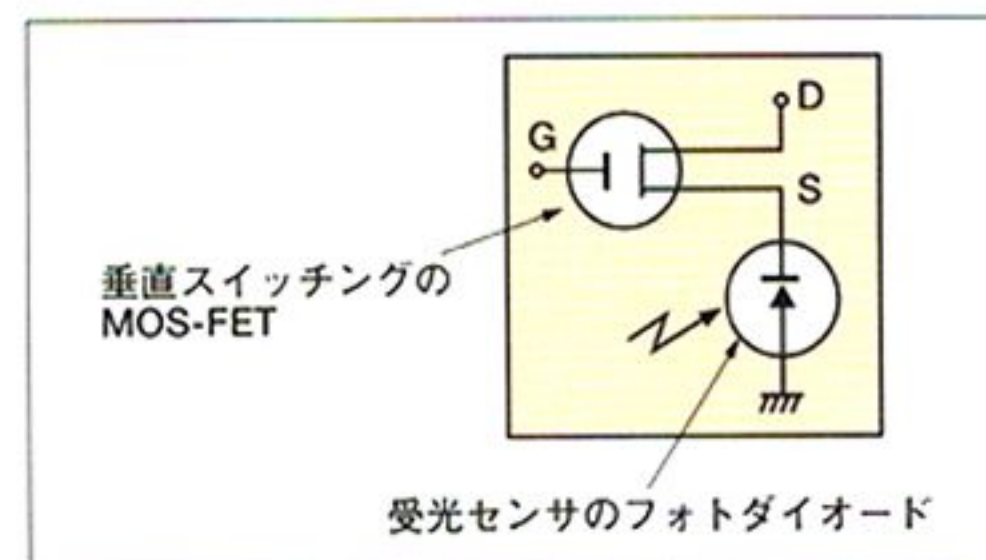
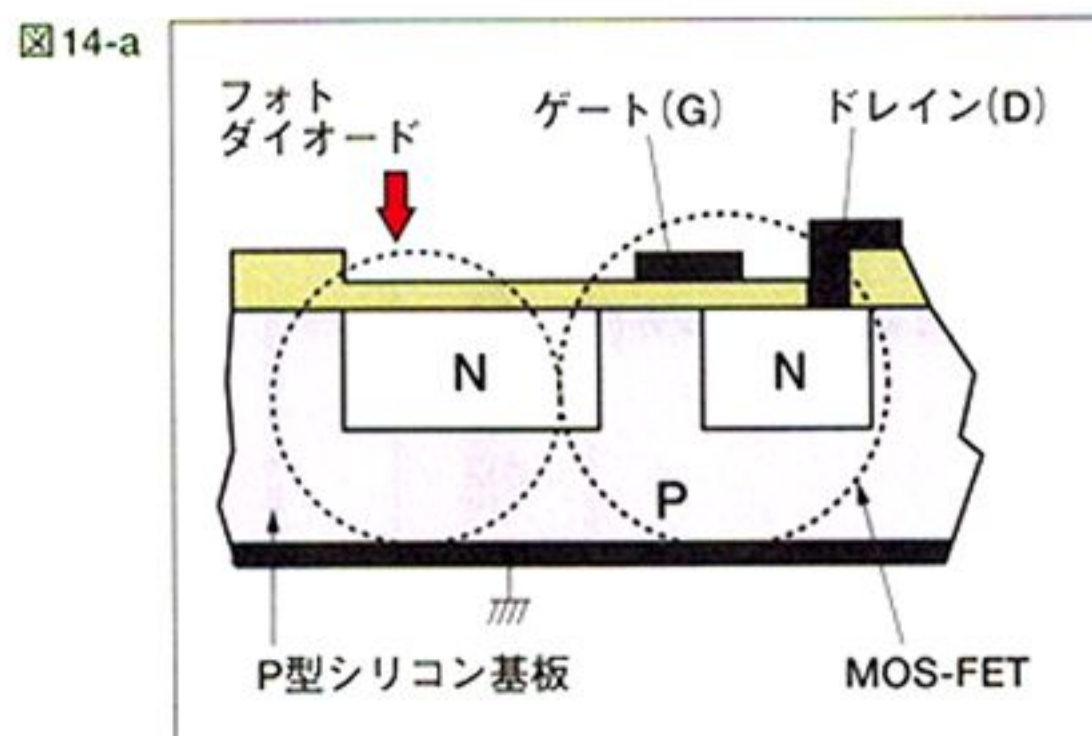
受光部もMOS-FETもちょうど，格子状に並んでいるわけだ。

そして垂直スイッチング用MOS-FETは垂直シフトレジスタに，水平スイッチング用MOS-FETは水平シフトレジスタに接続されている。シフトレジスタとは入力パルス(トリガ)を受けるとオンになる場所が順次移っていく回路のことだ。

ここまでの話を概念図にすると図15のようになる。

MOS/CMOS型イメージセンサの動作概念

さて，いま，受光部が光を受け，それぞれの受光部で光量に応じた電子が作られたとしよう。これがどのように取り出されていくかを見ていくことにする。

図16を見てほしい。

まずはこのA行目の水平線に電気を流す。この電気は受光部に取り付けられた垂直スイッチング用のMOS-FETに到達して，このすべてがONになる。ただし，この垂直スイッチング用MOS-FETは水平スイッチング用MOS-FETに繋がっていて，ここがまだ通電していないので，水平スイッチング用MOS-FETはすべてOFFの状態だ。よってこの時点では出力ラインからはなにも出てこない。

ここで，今度は1列目を通電してやる。すると水平スイッチング用MOS-FETがONになるので，この瞬間に受光部「ア」の電子流が出力ラインから出てくるのだ(図17)。

CCDのところでも解説したが，受光部では光がPN接合付近に到達すると光電効果で電子と正孔ができ，電子はN型，正孔はP型のほうに引き寄せられる。MOS-FETのスイッチがONになったときにここへ電気が到達すると，この状態を元に戻すように充電が行われ，このときの充電電流としての電流量こそが，この受光部の出力に相当している。

このあとは，水平シフトレジスタにトリガを入れれば，1列目の通電が取りやめられ，今度は2列目が通電される。このときには受光部「イ」の出力が得られるわけだ。

そして同様に水平シフトレジスタにどんどんトリガしていけば，A行のデータがすべて取り出される。

今度は垂直シフトレジスタにトリガを入れて，B行目を通電，同じく水平シフトレジスタは1列目に戻ってこのラインを通電。今度はB行目の受光部の情報が取り出されることになる。

このように，座標軸のXとYを指定して1点ずつ受光部の情報を読み出す方式から「X-Yスイッチング」方式とか「X-Yアドレス」方式などと呼んだりする。このX-Yアドレス方式では，各受光部

図14-a

図14-b

図15

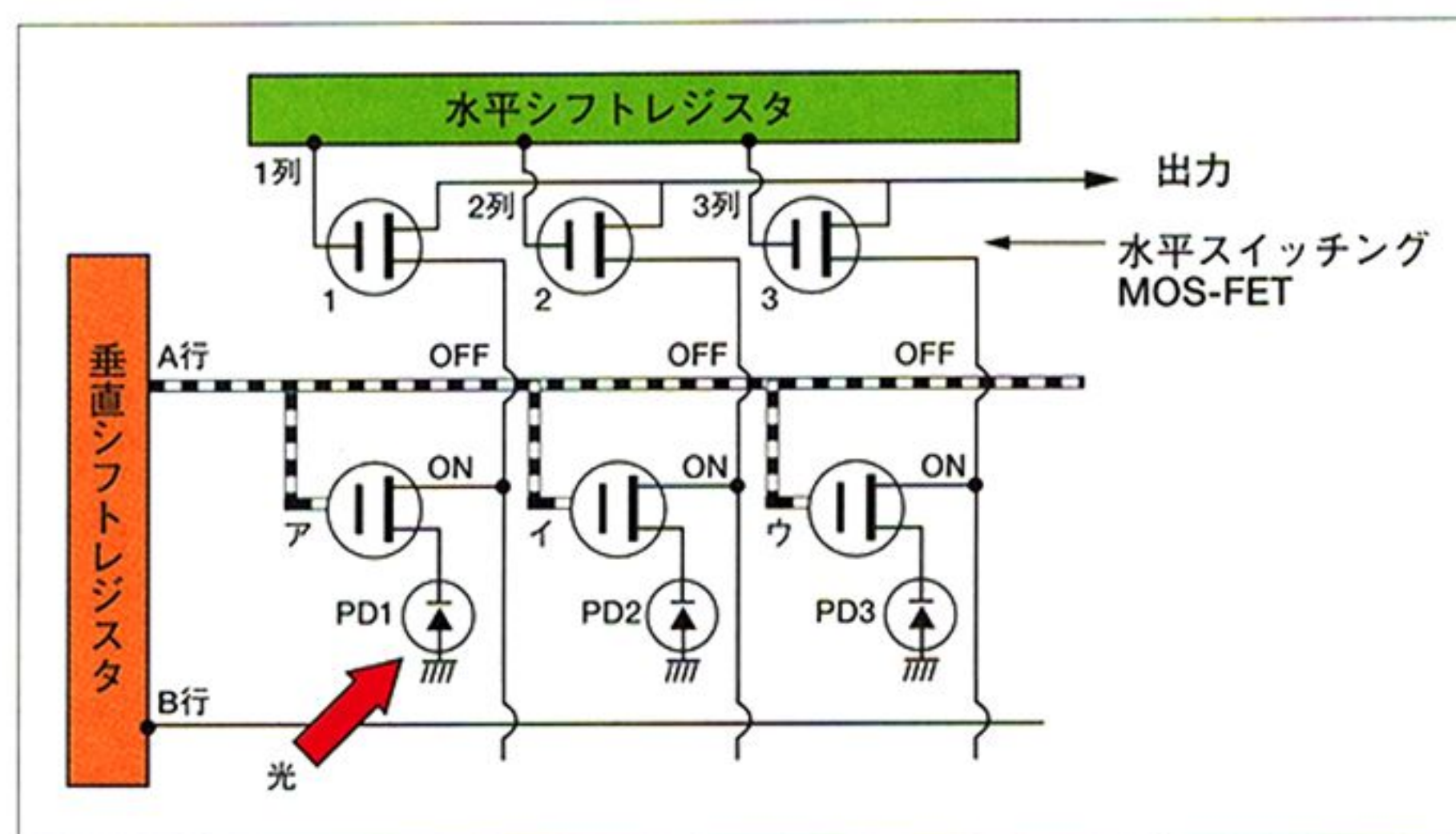



図16

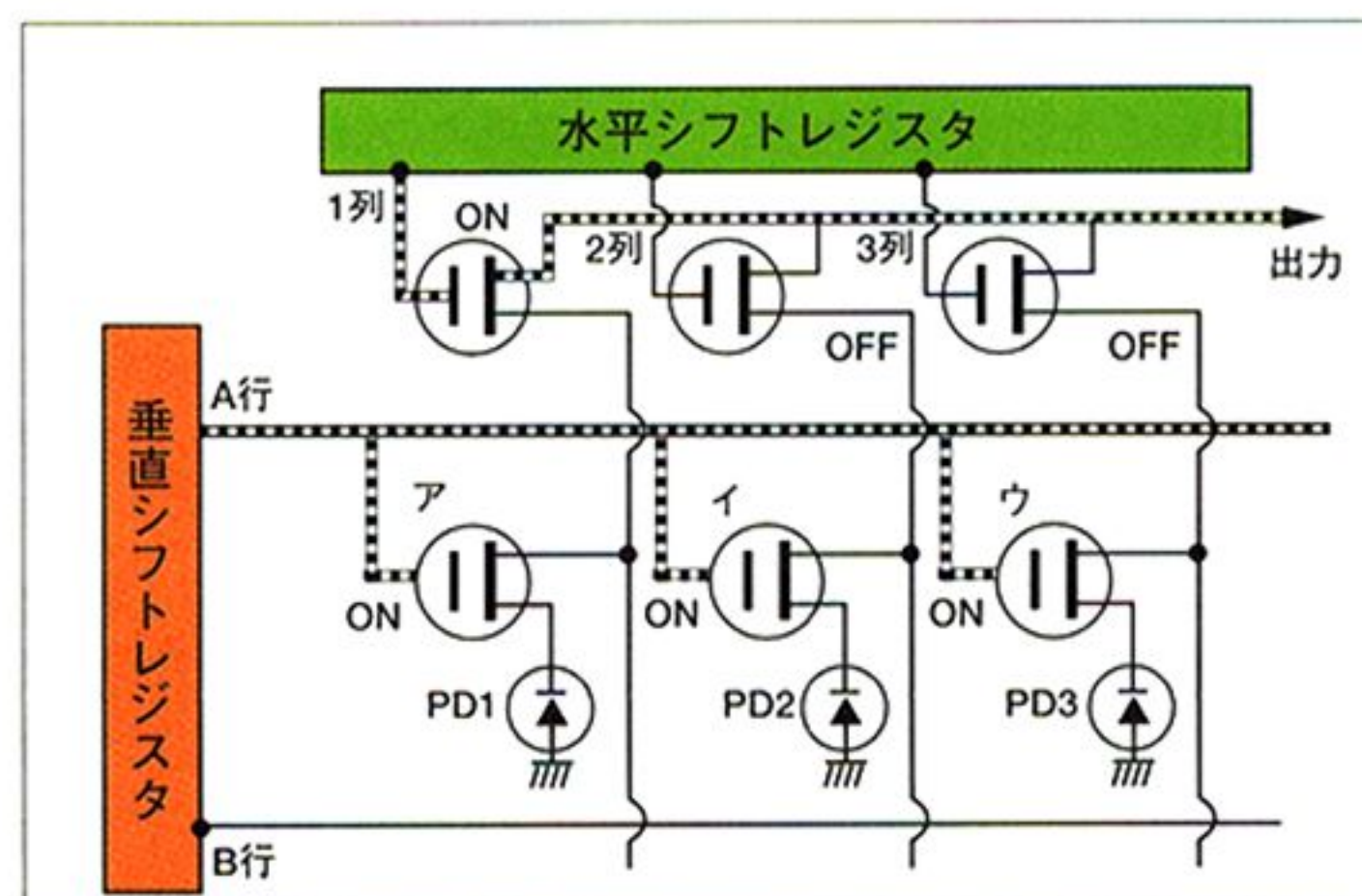



図17

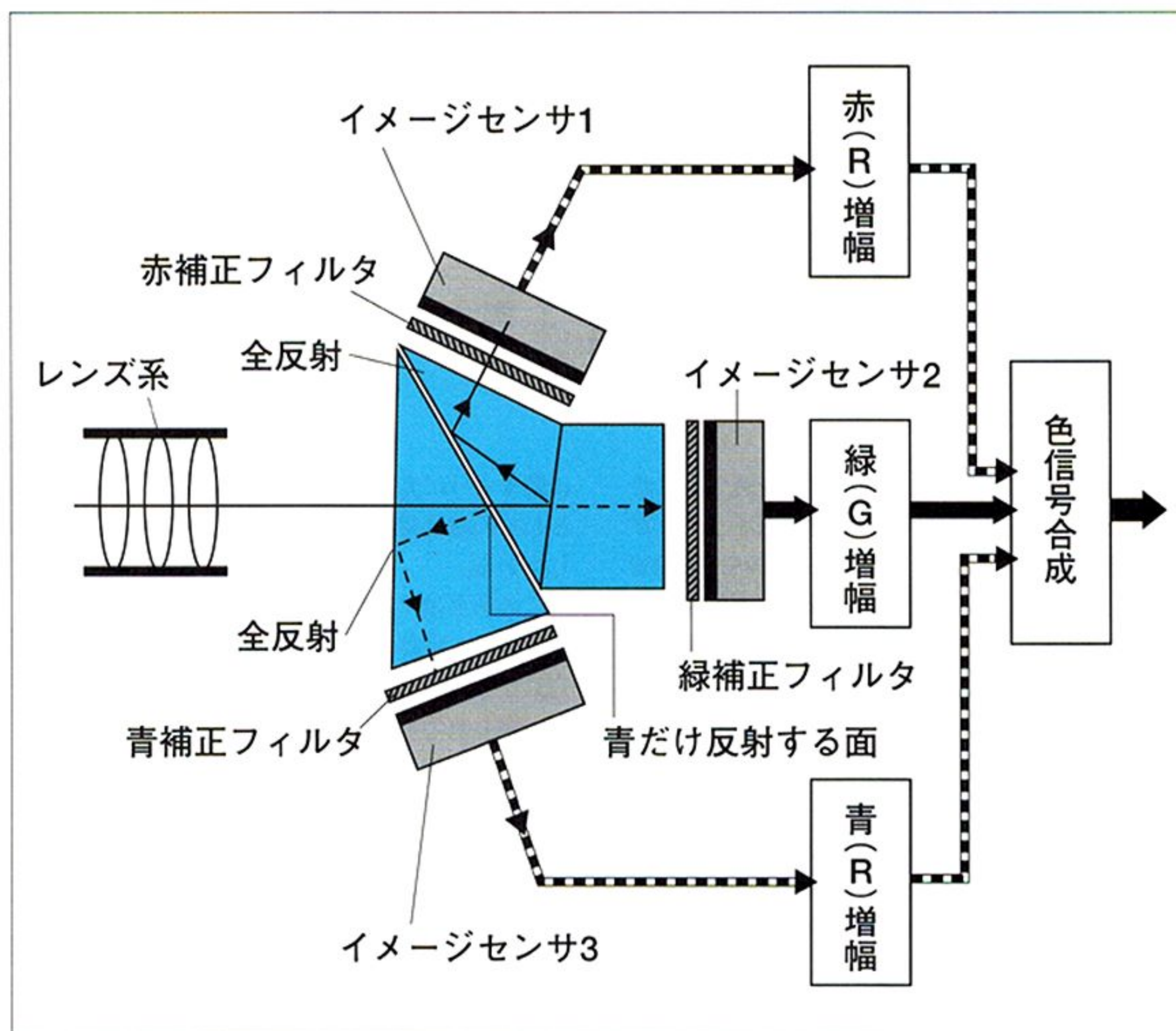



図18　3板式CCDカメラの構造

の情報が1つひとつ取り出されていくため，信号が取り出されるまでは，その受光部はずっと受光し続けてしまう。このためこの素子には電子シャッター機能はないことになり，デジカメの撮像素子として使う場合には機械式シャッターが必須となる。

MOS/CMOS型イメージセンサの現在過去未来

さて，ここまではMOS型についてのみの解説を行ってきたわけだが，CMOS型もFETにCMOSを使っているだけでほぼ同じものと考えていい。ただし各受光部で得た情報をその場で増幅する(セルアンプ)機構があるため，MOSよりもノイズの影響を受けにくいという特徴がある。

ところで，MOS型イメージセンサの発明自体は実はCCDよりも4年早い1966年に行われている。歴史はCCDよりも長いことになるのだが，実用素子としての主流はCCDにその立場を一歩譲った感じになっている。これはなぜだろうか。

その理由はいくつかあるが，まず，配線が複雑になるために受光部が小さくなり，感度が落ちるという欠点が指摘されたようだ。開発当初はCCDの半分程度の感度だったといわれている。

そして，この方式の信号読み出しの特徴であるスイッチング動作自体がノイズを発生させ，出力信号のS/N比が悪い点も弱点といわれた。

ところがこうした問題点は最近の半導体製造技術の進歩で過去のものになりつつある。

受光部の大きさについては，配線幅を細く，なおかつ受光部と立体的な構成にすることでCCDと同じ感度が実現できるようになり，ノイズの問題についても，そのノイズパターンが固定である点に着目し，ノイズキャンセル機構と組み合わせることでほぼ解決されている。

また，MOS/CMOS型の場合，スイッチング動作を速くしていけば，それだけ速く信号が取り出せるため高速化がしやすく，リアルタイム性が高くなるというメリットがあり，使用する製品のコンセプトによってはCCDよりも向いている場合もあるという。

さて，容易に想像されるとおり，CCDとMOSの両方の構造を組み合わせた撮像素子も続々開発されている。

そのひとつはCPD(Charge Priming Device)型だ。CPDでは受光部がMOS型と同じで，水平転送部をCCDで行っている。

水平転送をCCDにしたのは理由があって，MOS型のノイズの根元となったのが水平スイッチング動作にあったからだ。CCDは電線でなく，無接触転送だからノイズの影響は受けにくい。ここに着目したわけだ。

もうひとつはCID(Charge Injection Device)というものだ。受光部で得た電子を基板内に注入して伝送線に取り入れ，これを読み出していく。

このほかにもCSD(Charge Sweep Device)，BBD(Bucket Brigade Device)，CMD(Charge Modulation Device)，SIT(Static Induction Transistor)などなど，さまざまなアイデアを組み合わせた撮像素子が開発されている。

CCDを用いたカラーカメラの実現方式

CCDの仕組みや種類がだいたいわかった。面としての光の情報を取り出せるのもわかった。しかし，二次元的な光の強弱がわかっただけでは，ただの白黒画像が得られるだけだ。

ではカラー画像を得るにはどうしたらよいのだろうか。

3板式カラーカメラ(3CCDカメラ)

光の色というのは赤(Red)緑(Green)青(Blue)，すなわちRGBの3原色の混ざり具合で表現されている。入力光をこの原色に分割し，RGBそれぞれの光の強さをCCDでとらえ，これをデータとして得ることができれば，CCDを使ったカラー撮像システムを構成することができることになる。

このシステムのことを3板式カラーカメラ，あるいはRGBそれぞれの原色に対してCCDを使うことから3CCDカメラと呼ぶ(図18)。

この方式は，高画質なのだが，入力光を分割した3色光が座標位置的にぴったりあっていないと映像の品質がめちゃくちゃになってしまうため，それはそれはデリケートなシステムとなる。

光学的に完璧な設計でなければならないし，使用している途中でこの完成された光学システムが狂うことも許されないのだ。

また，入力光を3原色に分割する関係上，光を取り回す光学デザインが必要になるため，カメラとしてハード設計する場合，大きくなりがちになる，という欠点がある。

よって，この方式は放送用カメラ，業務用カメラに採用されてることが多い(*11)。

(*11) 3CCDのビデオカメラは最近では一般ユーザーの手の届くものになってきた。デジタルスチルカメラとしてこの方式を採用したものとしてはミノルタのRD-175くらいしかなく，採用例は少ない。

単板式カラーカメラ

ひとつのCCDを使うのが単板式カラーカメラだ。現在民生品として我々がよく店頭で見かけるデジカメのほとんどがこの方式だと思っていいだろう。

さて「1枚のCCDでは光の強弱しか感じられないのでは？」と思うはず。

そこで，単板式カラーカメラでは，この1枚のCCDの受光部の外側にカラーフィルタを張り付け，それぞれの受光部では特定の色の光量しか出力しないようにしている。これにより，各受光部は，自分のところにやってきた色の光量を出力す

るようになる。

実際のカラーフィルタは図19のようになっている。左側が「原色フィルタ」，右側が「補色フィルタ」と呼ばれるものだ。実際のカラーパターンの配列は各メーカーがさまざまな工夫を行っているのでこの図と必ずしも一致しないが，大別するとこの2種類となる。

いずれの場合も2×2の格子がひとまとまりになっているのがわかるはずだ。つまり，ある色の1ドットを表現するためには2×2の4ドットで表現するということになる。受光部の1つひとつはフィルタを通った色の光量しか量ってくれないので，そのCCDが出力するカラー解像度は計算上は受光部解像度の1/4になってしまうことになる。

これでは，たとえば200万画素のCCDで作られる画像の解像度は50万画素相当になってしまうことになるわけだ。もちろん，このままでもいいのだが，実際のデジカメ製品のほとんどでは，ある着目した画素の周辺の色情報から相関を導き出し，CCDのフル解像度の画像を合成している。

一般にデジカメ製品はCCDの解像度にだけ着目されがちだが，この「カラーフィルタの特性」「合成プロセス」も「人間が見た目の画質のよし悪し」に大きく影響する部分なのである。

●原色フィルタ

図の左側のカラーフィルタは「原色フィルタ」と呼ばれるものでCCDの受光部に光の3原色RGBを通すためのフィルタだ。これは「加法混色」という色彩理論に基づいて設計されたカラーフィルタだ(図20)。

2×2で構成される1ブロックに着目すると，GがRBの2倍あることに気づく。Gが多いのは緑がほかの原色に対して人間の目での視感度が高いため，最終的な画像としたときに解像度低下を抑える工夫でこのようにしている。

ストライプ状にRGBを並べた原色フィルタはないのか，という疑問も出るだろう。結論からいえばそういった原色フィルタを採用したCCDもある。また，Gのみをストライプ状に配置させる原色フィルタを採用した製品もあったりする。どうも，この「カラーフィルタ」という分野にも奥深い「その筋の世界」が広がっているようなので興味がある人は各自でリサーチをかけてみてほしい。

●補色フィルタ

図の右側のカラーフィルタは「補色フィルタ」と呼ばれるものでCCDの受光部に原色2色の混合光を通す。2色の混合光で色が表現できるのかという疑問が出るかもしれないが，これは「減法混色」という色彩理論に基づいているのだ(図21)。

Cは水色(Cyan)，Mは赤紫(Magenta)，Yは黄色(Yellow)だが，このフィルタもこのCMYを2×2を1ブロックとして並べて構成している。

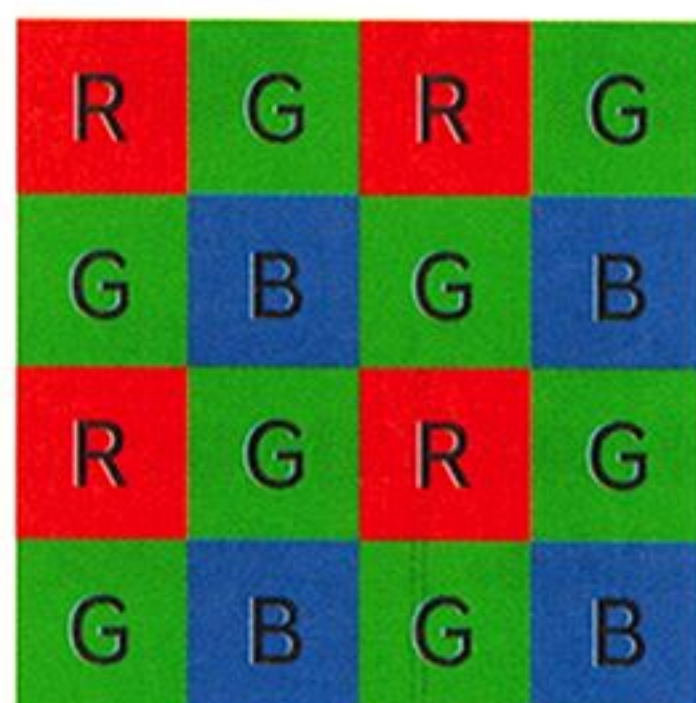
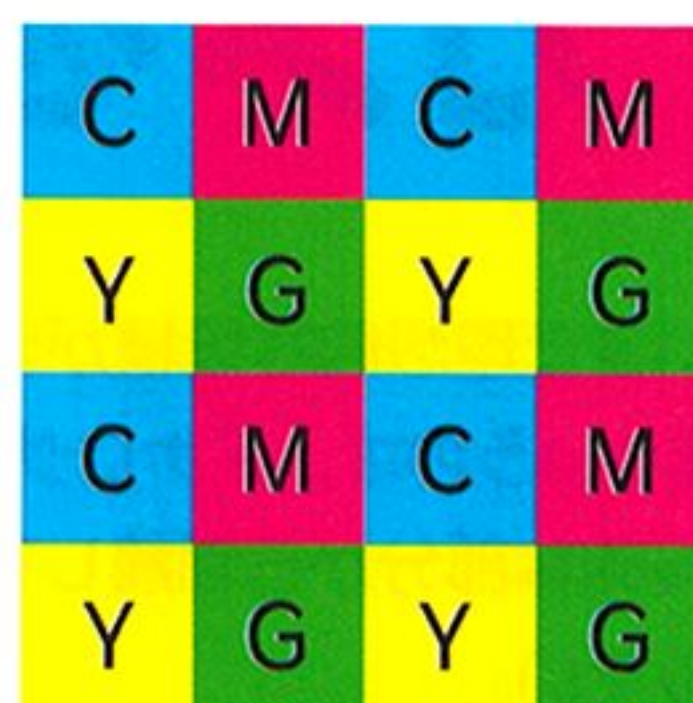
CはGとB，MはRとB，YはGとRの，2色の光を通すため，原色フィルタよりも多くの光がCCDの受光部に届くことになる。受光部に多くの光が届くということは高い感度が得られるということだ。これはつまり各受光部は2色分の光の明暗情報をとらえているため，画像の本質的な解像度も優れているといえる。

ところで，図に補色でない原色のGのフィルタが配置されているのは，やはり，視感度の高いGを重視するための工夫によるものとされている。

図19

おわりに

現在デジカメに採用されているCCDは200数十万画素のものが主流だ。今後さらに高解像度化される可能性は高いが，受光部が小さくなって高密度化すると感度が悪くなるという問題がある。また，CCDの受光面に投影する映像を作り出すための，レンズの光学的な解像度との兼ねあいもあり，今後はどういう方向性でデジカメ，CCDが進化していくかが非常に興味深い。

あるいは近い将来，CCDに変わるイメージセンサが開発される可能性もあるだろう。そのときは撮像管からCCDに移行したときのような劇的なムーブメントがこの世界に巻き起こるはずだ。

なにしろ，イメージセンサの歴史が始まってからまだ半世紀しかたっていない。しかも，その電子化，デジタル化の歴史はさらにその半分だ。

我々の，まだ見ぬ高画質を探し求める旅は，始まったばかりなのである。

参考文献一覧

・読売科学選書1「新しい電子の目 CCD」野口靖夫著　読売新聞社
・入門エレクトロニクス「エレクトロニクスを支える半導体の仲間たち」泉弘志著　誠文堂新光社
・「電子デバイス入門」桜庭一郎著　昭晃堂
・「解明・新物理」小口高著　文英堂
・DOS/V magazine 1997年9月15日号 p.206「画像入力デバイスのかなめCCDとCMOS」坪山博貴　ソフトバンク
・DOS/V magazine 1998年11月15日号 p.173「カタログを読み解くためのテクニカルターム集」佐藤イツキ　ソフトバンク
・安藤幸司氏のWeb「AnfoWorld(http://www.dango.ne.jp/anfowld/index.html)」

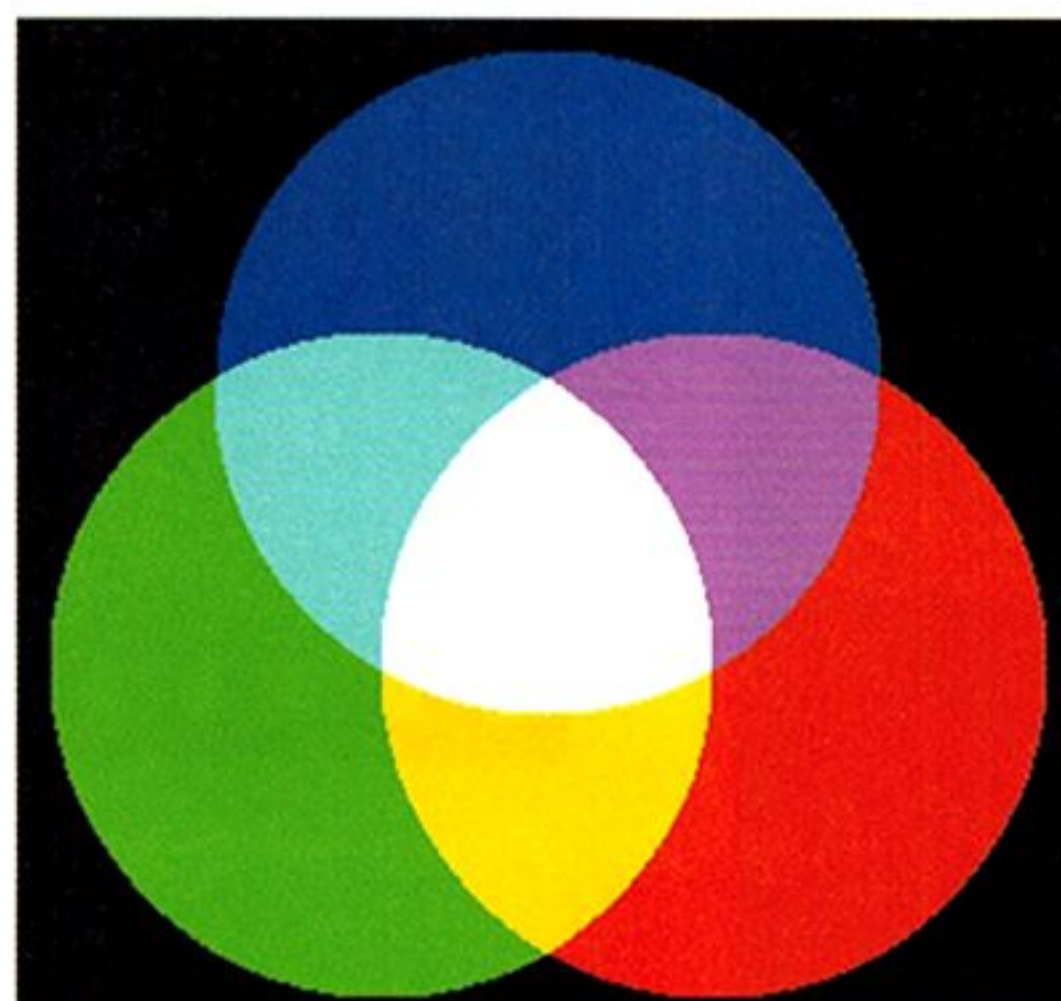



図20　原色フィルタはRGBを基本とする

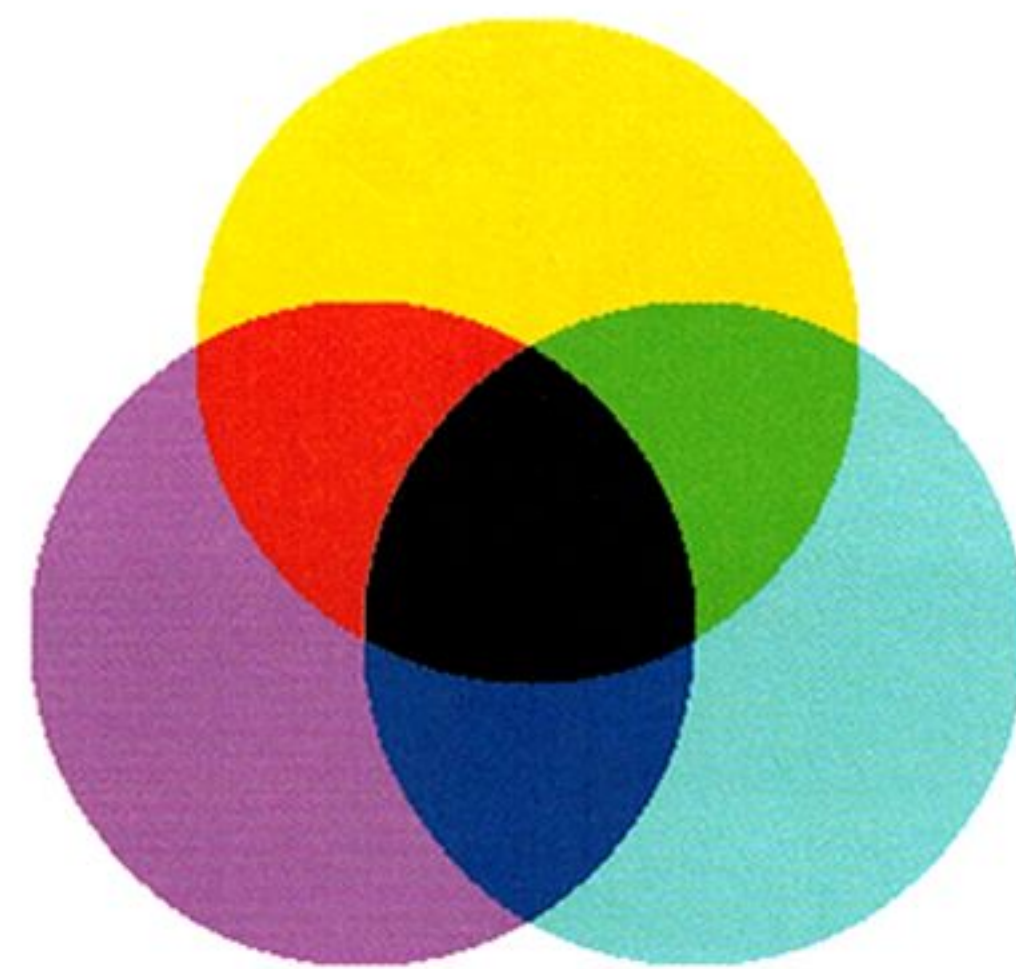



図21　補色フィルタはCMY系列だ

デジカメを改造する

中野修一 Nakano Shuichi

デジカメにはいまだ理想的な機種が現れていない，と感じている人は多いだろう。多くの人が「いけるかも」と思う機種というのは，アナログカメラとして見た場合に遜色ないものであることが大半だ。そういった判断は本質的な部分から乖離しているようにも思える。ここではあえて，1眼レフ式のアナログカメラを意識してみたい。

疑惑のデジタルカメラ

デジカメというのはデキの悪い機器だというのが率直な印象だった。

使い勝手が妙に悪い。もう1段開けたいんだけど……ということすら，素直にできない機種もある。一般のカメラやビデオカメラの操作性はおおむね素晴らしく使いやすい。どうしてこんなに多機能なのかと驚くくらいのものがシンプルな操作系に集約されている。日本の家電業界の実力からすれば，それが当然なのだ。

さらにデジカメは画質が悪かった（とりあえず過去形）。「ビデオ用のCCDだから……」などと書かれているとなにが問題なのかと頭の中が疑問符でいっぱいになったものだ。

画素が正方形でないから補間が必要，とはいうものの，VGAなどの場合，縦方向の解像度は同じと考えていい。NTSC CCIR601準拠フォーマットの場合，704（720）×480ドットで横解像度はむしろ高い。そういった場合に正方画素でないことのデメリットはなにも考えられない。ファームを組んでいるプログラマが足し算と引き算しかマスターしていないのならともかく，だ。どうせ複数画素から補間するのだから，演算誤差などのリスクは変わらない。ならば情報量は多いほうがむしろ有利だ。誤差が出たとしても8ビット中の1ビットくらいで（プラスマイナス0.75ビットくらいか？），これはノイズを考えれば十分無視できる。

「RGBに変換されるので補色系は」云々などといわれても，情報量自体は少なくとも減ることはない。印刷用のCMYとの変換が正確にできないことは有名だが，この場合は光学的な問題だけなので，数式どおりの処理でよいはずなのだ。情報量は変わりようがない。

また，RGBというものを強調されても記録をJPEGでやる以上，データはRGBではなくYIQに変換されているはずだし，なにより，JPEGフォーマットで保存するということ自体が，現在のDVカメラとまったく同じ処理方法なのだ。なのに，パソコンだとRGBだから……といわれる。なぜ？　と詳しそうな人に聞いても納得できる答えが返ってきたことはない。無理はいわないので，DVと同じくらいの色を出してくれればそれでいい。補色系フィルタを使ったDVで色が悪いとかいう話はついぞ聞いたことはないのだから。室内で青く色かぶりするなんてのをなくしてほしい。以前ビデオサロン誌がさまざまな照明でビデオカメラのテストをしていたことがあったのだが，少なくとも蛍光灯下で色かぶりするような製品は皆無だった。そもそもそれをクリアできないようなビデオカメラは商品として成り立たないというのだ。それはそうだろうと思う。

とりあえず作る

ということで，納得できない以上，デジカメに関して世間一般でいわれていることは鵜呑みにするわけにはいかない。ものごとにはしかるべき根拠があるはずなのだが，それが見えてこないのも気持ち悪い。だいたい「CMYGの配列です」といわれて，なぜ納得できるのか？　YIQに変換することが主目的であれば，Y（輝度）重視は当然のことだろうが，ならばCMYとスルーであるべきではないのか？　緑の値だけは正確に知りたい，とするのはほかに理由があるはずなのだ。

さてさて，デジカメの評価はアナログカメラの画調と比べて云々されることが多いのだが，じゃあ実際に同じような条件にしてやったらどうなのか，と試してみようとはしない。縮小光学系なので，人間のような大きなものを撮っていては同じように写るはずはない。そんなに一眼レフカメラの性能がほしいのなら作ってみればいいのに……と文句だけをいっていてもしかたないのでレンズ周りの問題について評価するためのデジカメ改造を行ってみた。

本来，デジカメの評価は「アナログカメラとは異なるのでダメ」という理不尽な視点で評価されるべきではない。メディアとしての可能性はさらに広がっているのだが，デジカメの位置づけとし

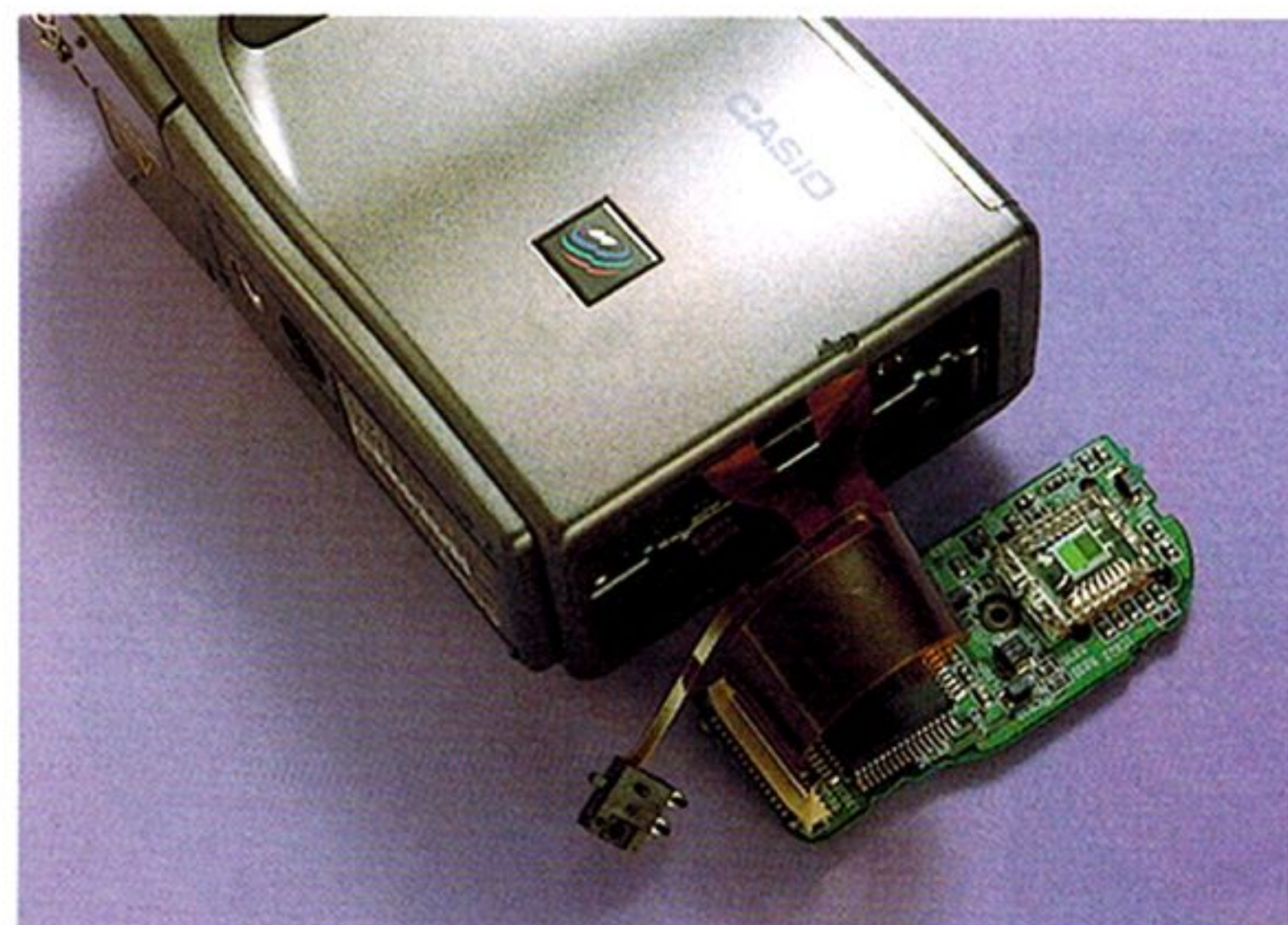



図1　QV-10からCCDからCCD部分を取り出してみた

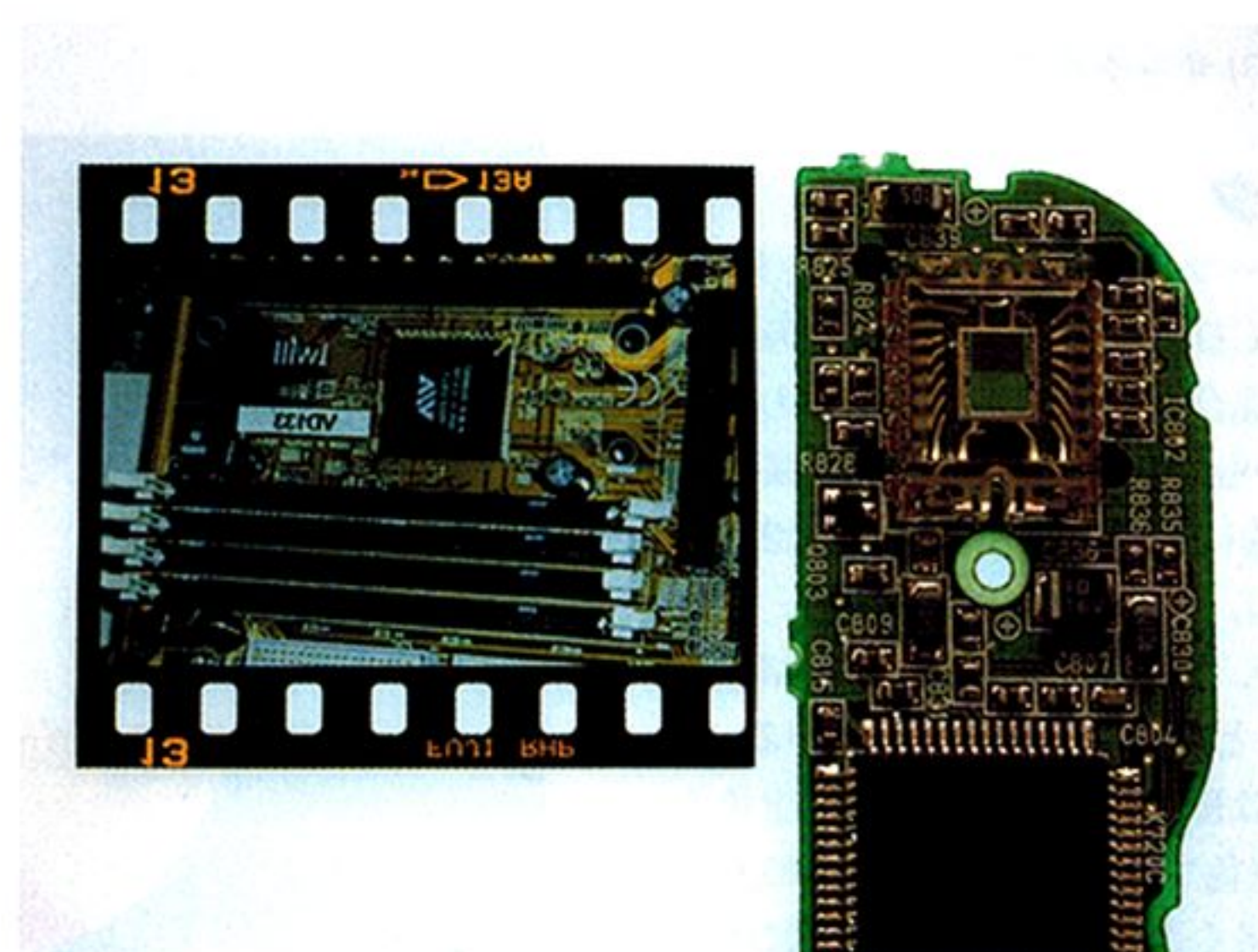



図2　QV-10のCCDと35mmフィルムとの対比

てアナログカメラの代替製品というのもないわけではない。特にアナログカメラメーカーはそう思っているのかもしれない。そのあたりが歪みの始まりのような気もするな。

ま，とりあえず，やってみよう。

実験：QV-10の改造

まずは，多くの人がどこかに転がしている可能性が高いカシオQV-10である。レンズ部分はあまり役に立ちそうにないので外してしまい，外部レンズが直接CCD部分に像を結ぶような配置にしてやろう。理論上はこれでちゃんと絵が撮れるはずである。

QV-10は非常によくできたデジカメだった。惜しむらくは画質が非常に悪い。現在の画質解像度絶対的な風潮では許されない機種ではある。

それでもデータを自前で綺麗に処理しようとするオンラインソフトのおかげでずいぶん助けられていたように思われる。

CCDの生データやフィルタ特性データなどが公開されたら色分解をやってみようという輩は少なくないはずだ。QV-10はメーカー製のソフトだけでは実質使いものにならなかったのだが，同様なことはどの機種でもいえそうだ。デジカメ内部のプロセッサでできないことでも，PCのプログラムならさらに適切に処理できることはあるだろう。そういう部分こそオープンにすべきだと思うのだが。

さて，レンズには2つのサイドがある。片側は長焦点，もう片側は短焦点になるように設計されている。一眼レフカメラの場合には，ボディ内のフィルム面で焦点を結ぶようにレンズが設計されている。このとき想定されている35mmフィルムとCCDでは面積に格段の違いがある。要するに，そのCCDの大きさで切り出した部分しか写らないということだ。大きさの違いは具体的には図2のとおりだ。QV-10の特に小さいCCDでは，卒業式の集合写真を撮ったつもりでも仕上がりはツーショット写真になっているといった感じだろう。超望遠のデジカメがほしいという人以外はあまりうれしくないだろう（デジカメで月面写真を撮るとか）。

これではちょっと使いづらいので（使いものにならないともいう），CCDの手前で映像を縮小ということで，CCDの前面に高倍率のレンズを入れてみる。

「そんなことをして大丈夫なのか？」

という疑問はもっともだ。基本的にちゃんと縮小された画像が投影される。レンズの色収差など，よくない影響は発生すると思われるが，ちゃんとしたレンズ設計ができるわけではないので，ここは，

「結果を見て判断」

ということで割り切りたいと思う。原理上は，CCD面積自体を大きくするか適切な倍率の適切なレンズを置くしか手がない。「そもそも置いちゃいけない」という選択肢を認めると話が成り立たなくなるので，強行しよう。

図3　レンズをつけた（ケース側）

図4　マウント各種（キヤノン，ニコン，ミノルタ）

で，35mmレンズをどうやって固定するか？　こういったレンズはカメラボディのマウント金具に固定するようになっている。きちんとマウントしないと話にならない。これにはレンズキャップを使おう。レンズのマウントと同じ機構で固定でき，マウント部分のみを使って，穴を開けてやれば，ほぼ望みどおりのものができる。

また，マウントキャップごと取り替えれば，各社のあらゆるレンズが使用できるようになる可能性がある。必ずしもできるとはいわない。たとえば，ミノルタαマウントレンズはマウント部のレバーを操作しないと絞りが閉まってしまうし，キヤノンは逆に開いたままになってしまう。ニコンMF用レンズは手動で操作できるので，今回はニコンAIマウントのものを使用している。キャップだけは用意したので，多少いじればほかのレンズでもある程度いけるかもしれない（実験はしていない）。

図5　できあがり

図6　背面図。市販のプラケースがぴったりだった

搭載するレンズは，できるだけ広角のものが望ましいことはいうまでもない。ただでさえ超望遠になりそうなのだ。機能面から見ると，できればズームレンズがいい。固定倍率のデジカメがズームつきになってこそ，使いでもあるというものだ。今回はシグマの18～35mmという広角系ズームレンズを使用した。最終的な画角からすれば，中望遠から望遠という感じで使えるものになる。

一眼レフレンズを求める人の大半がほしがっている部分にどんぴしゃな感じなのでこの辺に決めよう。

改造の実際

では，レンズ部を開けてCCDを取り出す。そのままだと固定しづらいので結局本体側も開けてフレキシ基板とCCD部だけを残して分解する。

QV-10のCCDは対角線で1/4インチしかない。見ると本当に小さいことがわかる。フィルム1枚分に絵を作るためのレンズで，この領域だけを使って絵を作っていくわけだ。そのままではどうしようもないのでレンズをつけて，先ほども述べたように画像を集約する。

今回は2つの両凸レンズを使った。両方ともかなり高倍率のものだ。

ちなみに，この倍率というのはなんだろうか。

人間は普通にものを見るときには25cmくらい離して見ることが多い（そうだ）。

250mmの焦点距離，これを等倍として，焦点距離の比を表したのがいわゆる「倍率」となる。25mmなら10倍，100mmのレンズなら2.5倍だ。

レンズというのは，まともに写るように設計されている。本来はいじらないほうがいい。あえてという場合，メーカー保証は捨てる覚悟が必要だ。

デジカメというのは，あまり中を開けていいも

のではない。精密機械であることはもちろん，なかにはあまりバラすということを考えずにあちこちハンダづけで処理してしまっているものもあるのでものによってはバラせない可能性もある。

ストロボつき機種の場合は特に気をつけないと危険だ。電池を外してもコンデンサに電荷が残っているので，分解時には十分気をつけなければならない。ショートするとほぼ確実にデジカメ本体が壊れてしまう。念のためにいっておくと，本来分解したりすべきでないのはいうまでもない。開けた場合はメーカー修理はあきらめよう。こういうことをしていると，気をつけていても壊れるときは壊れるものだ。万一，同様な方法を試すときには必ず「壊れてもいいカメラ」を使用すること。いや，実際，あっさり壊れるものだ。

最初にコンデンサ部のラインをシールドするなど，事故防止にはいくら気をつけてもつけすぎということはない。

とにかく，CCDをむき出し状態にできたら，動作テストをしなければならない。この状態でデジカメとして機能するかどうかである。QV-10の場合は問題なく動作した。色はちょっとおかしい（もともとか？）。パンフォーカスだったものが，レンズの開放値がミニマムになったことで，被写界深度が極端に浅いことも確認できた。ということは，これは結構いけるかもしれないということになる。この段階ではとにかく動作することが重要だ。

CCDを適当な位置に固定するのは難しいのだが（距離をきっちり決めないとフォーカスがあわない），ある程度の微調整はレンズ側のフォーカスリングで対応できる位置にしっかり固定したい。レンズ2個を使ってかなり派手に光線を絞っている。CCDの大きさは1/4インチというコンパクトさなので性能は期待できない。レンズを変えてよくなったか？　というと，被写界深度が浅くなったぶんだけで画質アップしたというわけにもいかないだろう。

FinePix500の改造

さて，少し欲が出てきたので，もうちょっとマシなCCDを持ったカメラで試してみよう。

いちばん安く手に入るデジカメということで引っかかってきたのが富士写真フイルムのFinePix500だった。中古を除けばダントツで安そうだった。性能自体はほぼ問題がない。ズームがついていないのも，むしろ好ましい（こういう用途には，だが）。CCDは1/2インチの150万画素のものを搭載している。

同様にレンズを外して，動作を確認すると，オートフォーカスやシャッターなどがなくてもちゃんと信号は出てきて，記録されるようだ。これならなんとかなる。

今回CCDへの光線集中用に使用予定だったアクロマチックレンズ（いわく「アクロマチックレンズは分散の低い材料（クラウンガラス）の正のレンズと分散の高い材料（フリントガラス）の負のレンズを組み合わせ，異なる波長の光に対して色収差を除去します。単レンズに比べて高い透過率，像の明るさ，高分解能を持ち，球面収差に対しても単レンズよりはるかに改良されています」だそうだ）だが，今回，口径にあわせて入手したものは，分厚すぎて導入を見合わせざるをえなかったが，色ムラなどが多く発生するようならこのようなレンズを使うことを考えていくべきだろう。

CCDを所定（？）の位置に固定してレンズをつけてしまえば，意外と簡単に作業は終わってしまう。

画質比較

「レンズの解像度自体の問題はデジカメのほうがいいんだ」といういい方はたまにされるようだがたいていの人はそこまでは気にしない。多少ぼけていたりするものは縮小するなりすればほぼ問題がなくなる。

小さな液晶でマニュアルフォーカスすることになり，ちょっとつらい感じ。デジタルズームでフォーカスをあわせ，引いて撮影した。それでもまだまだフォーカスは甘い。被写界深度は浅い。

撮影はマニュアルモードで行った。

どうしても収まりが悪かったので青いフィルタガラスを外したため全体に色がかかっている（致命的？）。カラーバランスはオート設定だとさらに赤くなる。これは日中モードで撮影したものなのだが。

フォーカスが全域であっていないのはともかく，妙に横縞が見え，画像が粗い。横縞の原因はなんとなく予想がついている。メカニカルシャッター機構を抜いているので，フィールドごとに増幅率が違うのだろう。これは1ラインごとにやや明るめになっていると考えていいので，奇数ライン偶数ラインごとの明るさをフィルタ処理で調整してやるのが適当だろう。

懸案の（？）被写界深度は十分狭くできる。もうちょっときっちり作れば，一眼レフカメラとほとんど変わらないボケ味などを出すことも十分可能だと思われる。光学系の利用という意味ではだいたい予想どおりの結果になってきている。

最後に

残念ながら，今回の改造はあまり成功したとはいいがたいものがある。結果で示すことがもっとも必要だっただけに，細かい詰めが間に合わなかったというのは残念だ。フィルタを取りつけ，後処理用のプログラムなどを加えて初めてシステムとしてかたちになるものだろうと思う。

今回ベースで使用する機種の選択にてこずったが，

CCDがなるべく大きく
なるべく安く
レンズなどを取り除いても動作する

というのがポイントになった。レンズは用意するわけだから，ズームつきなどに意味はない（分解が難しくなるだけ）。多少古い機種が箪笥の肥やしになっているという場合などは手持ちのものを試すのもよいだろう。

なんやかんやでずいぶん高くつく記事になってしまったが，カメラのレンズなどは中古レンズを探してくれば，さほど高価なものではない。今回使ったのは18mmからの広角系ズームというレアなものだったが，それでも2万円ちょっと。単体レンズを買うと1個数千円（アクロマチックレンズはさらに高い）だから，それを考えると案外安

図7　これが FinePix500

図8　FinePix500のレンズ部。今回は使わないので取り外す

図9　CCDの大きさをQV-10（右）と比べてみた。こんなに違う

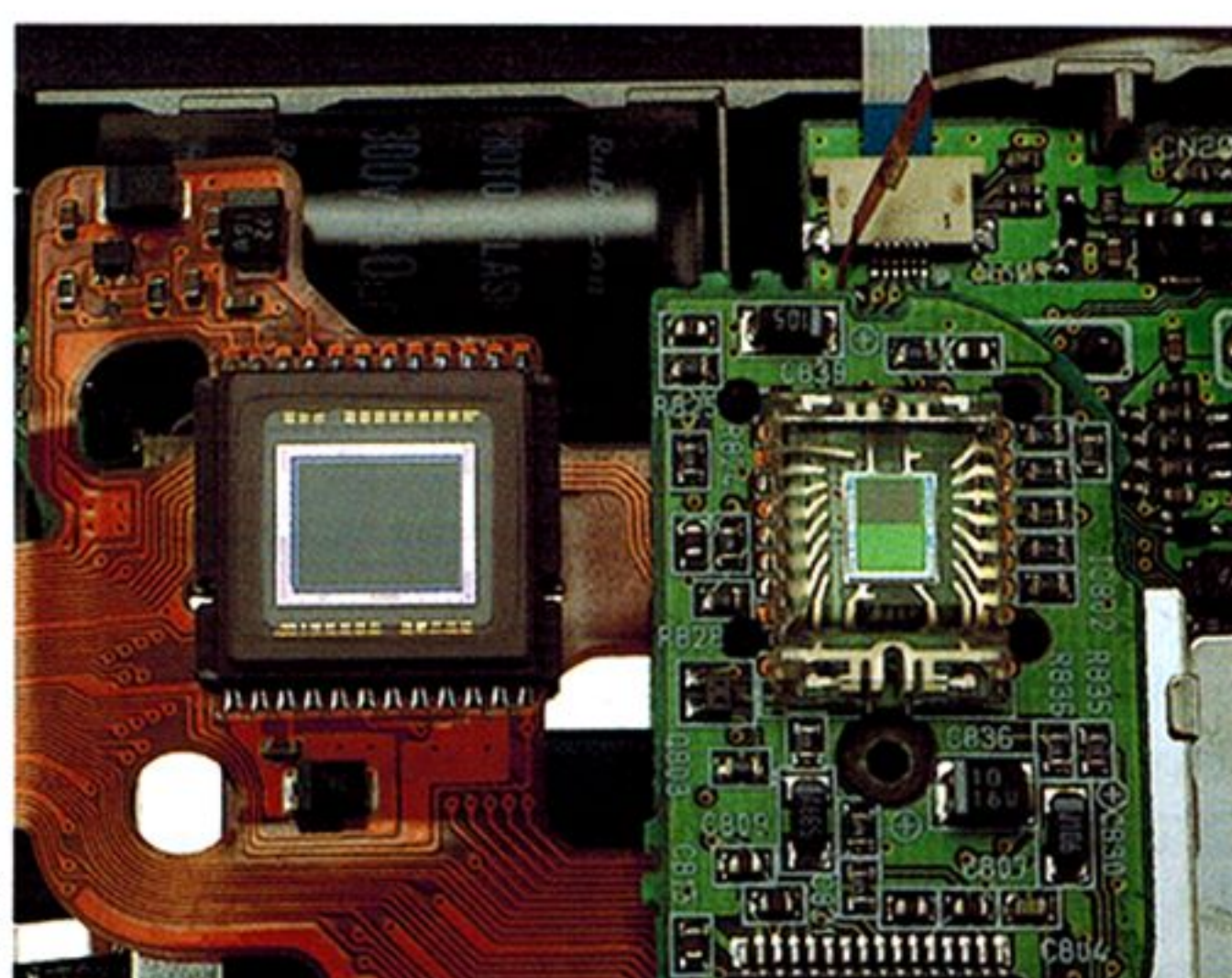



図10　だいたいこんな感じでマウントを取りつける

図11　厚すぎて今回は使えなかったアクロマチックレンズ

図12　レンズをCCDのできるだけ近くに入れた

図13　できあがり。隙間部分は黒い布などでシールすること

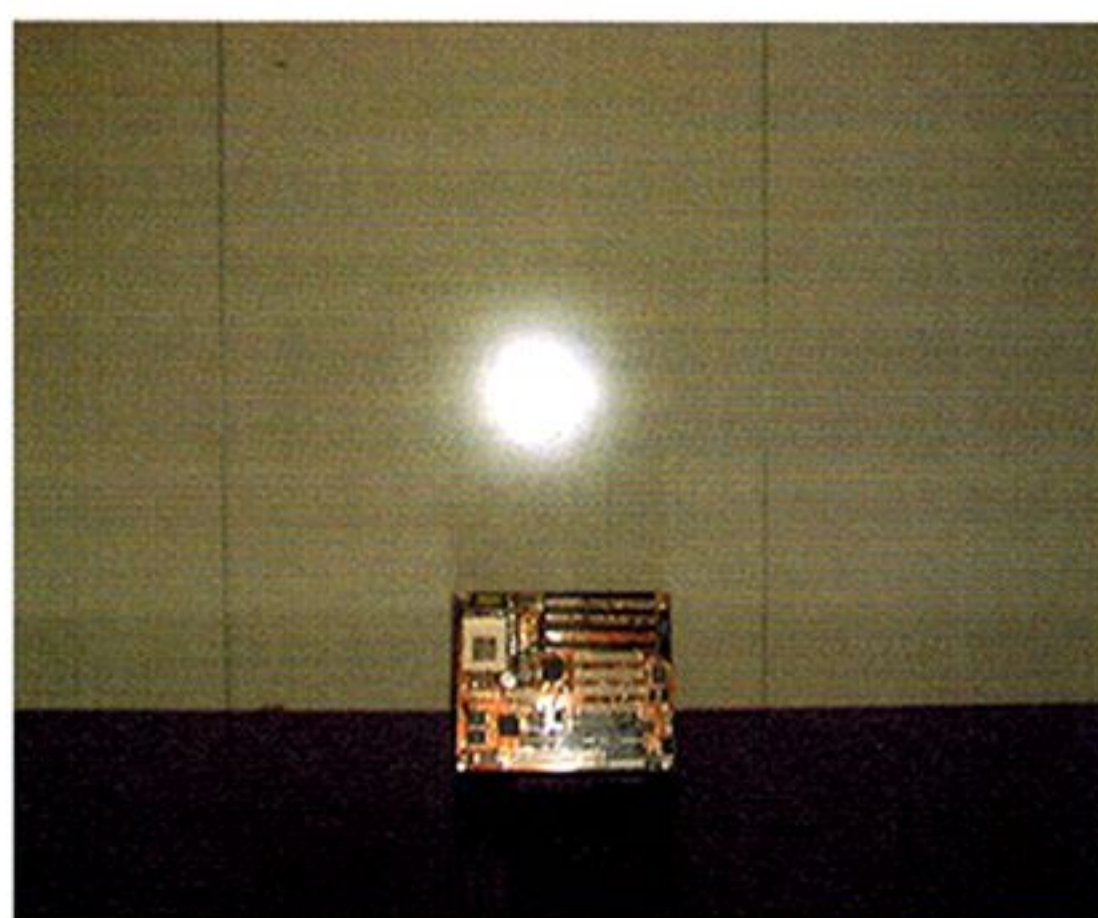



図14　この画角がオリジナルのもの

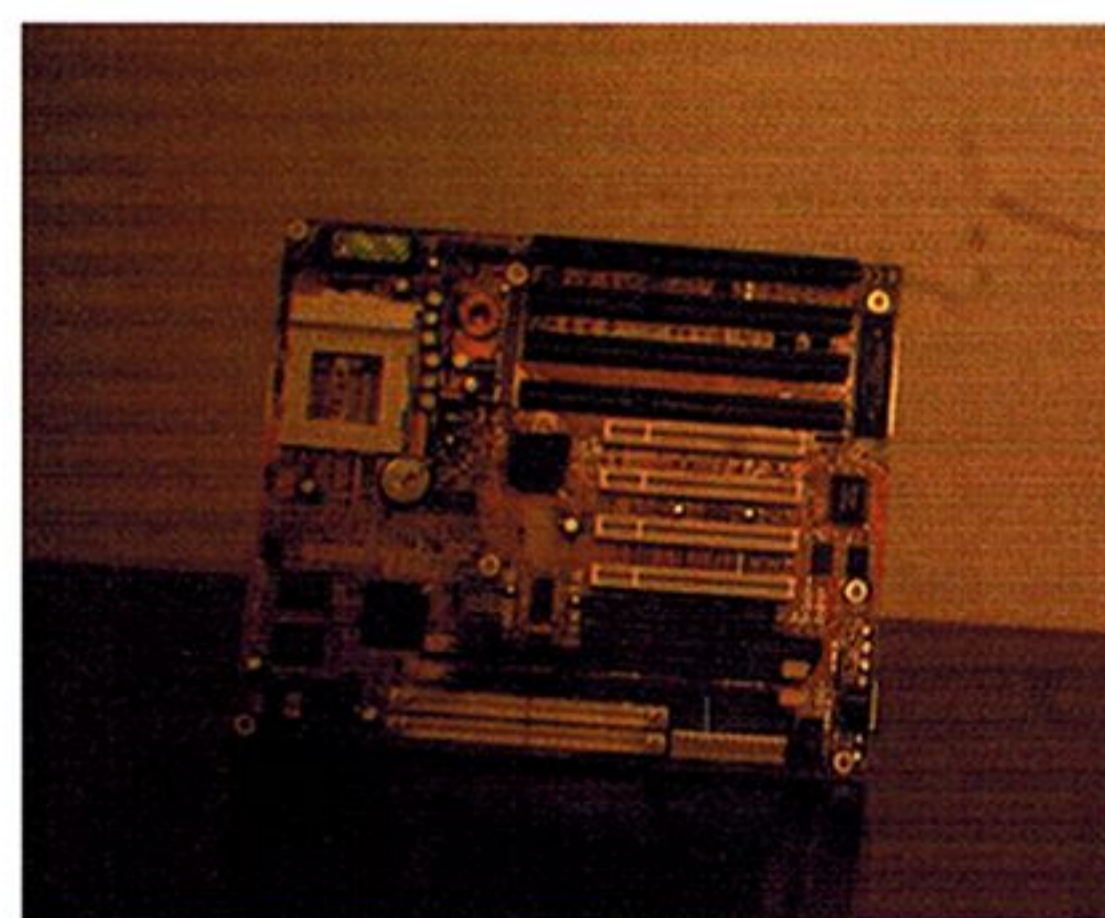



図15　最大望遠にしたときのもの

いといえるのかもしれない。

今回はなるべく元に戻せないような極端な改造は避けたのだが，そのために一部不備となる部分もあった（フィルタ関係など）。それにしても買ってきたプラケースと2機種がそれぞれかなりよくあった大きさだったのがよかった。ケースの内部突起部などは削ったのだがわりとよい感じで収まった。ただ，キャップに穴を開けるためのボール盤など多少の工具は必要だ。機会があればもうちょっと詰めた設定でチャレンジしてもらいたい。

＊

それにしても，現状ではデジカメの画質もかなり上がってきており，使い方自体は通常のカメラと変わらなくなってきた。撮影テクニックなどもアナログカメラの初歩の初歩が繰り返されているといった感じだ。

現状の最新型機では，写りの違いがあっても，カメラの違いは個性と考えてそれを踏まえたうえで使いこなしていくようなことを考えるべきだろう。それはアナログカメラと変わらないものであり，感性の部分でしか説明できない部分なのでもはやこの本の範囲ではないのかもしれない。

しかし，よくなったとはいえ，まだまだ改善すべき点も多い。

フォーカス特性は任せておくと前ピンになったり後ピンになったりするのだが，液晶画面ではきっちりあわせることがしにくい。CCDの特性上，任意部分の拡大など，読み出し順を変えるのは難しいのはわかっているのだが，フォーカス時に注視点をとことん拡大してフォーカスがあっているか確認できるような機構はついていてもよいのではないかと思う。読み飛ばしだけなら不可能でもなんでもないだろうし。

また，液晶画面の忠実度が疑わしい。撮った瞬間に確認できるということがデジカメの価値を大きく上げているのだから，きちんと確認できるような保証がほしいところだ。

3Dデジカメ？

ミノルタ3D 1500はミノルタとメタクリエーションズが組んで開発している3Dスキャンが可能なデジカメだ。150万画素のデジカメをベースにMetaFlash技術により，3Dデータを得るというもの。出たら即買いなのだが，問い合わせてみてもあまり気合が入ってなくて，資料はほとんどありません，技術的なお話もできません，といったなげやりな態度が印象的だった。

方式は詳しくわからないが，ストロボの光線角度かなにかで距離を計測するのではないかと思われる。

デジカメというのは，単に写真を撮るというだけではなく，こういったデータ入力デバイスとしてさまざまに活用していけるというところに本来の意味があるような気がする。

お気楽写真スキャン

鈴木典雄 Suzuki Norio

手軽な画像デバイスとしてデジカメは広く普及している。画像入力デバイスにはスキャナやフィルムスキャナなど，用途に応じてほかにもいろいろな種類がある。デジカメでそれらの代わりはできないものだろうか？　映像入力機器としてのデジカメのちょっと変わった活用法を紹介する。

「いい写真が撮れたからWebに載せたい」「友達に電子メールで送りたい」「年賀状にしたい」……誰でも一度や二度，こんなふうに思ったことがあるに違いない。そういうとき，近くにスキャナがあれば簡単に目的は達成できるが，なければPhotoCDに代表される写真屋さんのメディア変換サービスに頼らなければならない。しかし，印刷原稿にするのならまだしも，電子メールで友達にちょろっと送るためにメディア変換サービスを使うのは大袈裟である。「あのとき，デジカメで撮影していれば」……そう後悔しているあなたのために，お手軽な写真のスキャン方法を紹介しよう。

最初に断っておくと，これから紹介する方法はすべてデジカメが必須である。といえば，もう勘のいい方はだいたいわかったと思うが，要するに「デジカメをスキャナの代わりに使ってしまおう」ということである。使用するデジタルカメラは画素数が大きく，マクロ（接写）撮影ができ，さらにズームがついていれば文句なしであるが，VGAサイズで最短撮影距離約20cm，ズームなしといった2～3世代前のデジカメでも，Webに載せる，電子メールで送る，ハガキのスミにちょこっと載せるといった程度なら十分実用になる。

初級編
写真（プリント）からのスキャン

まず，簡単なところで普通の写真（プリント）からのスキャンをやってみよう。というと難しく聞こえるが，要するに写真を適当なところに置いて，パチッと撮るだけである。マクロ機能がついているデジカメの場合は結構近くまで寄れるため，写真が小さすぎてしまうということはないだろう。マクロ機能がないカメラの場合は，ズームや撮影距離をいろいろ変えてみて希望の大きさになる組み合わせを探ってほしい。

画素数が少なく，ズームもマクロもないデジカメの場合は，どうしても希望の範囲が撮影領域いっぱいにならないことがある。そういうときは素直にあきらめて撮影後にフォトレタッチソフトでトリミングする。こうなると印刷に使うのはちょっと難しくなるが，Webや電子メールで送る程度だったら十分実用になるだろう。写真を大きく焼き直せば問題は解決するが，枚数によってはまとめてPhotoCDに焼いてもらったほうが安かったりすることもありえるので，よく考えること。

昔，絹目の写真が流行ったことがあるが，絹目よりも光沢仕上げの写真のほうが綺麗にスキャン（？）できる。ただ，光沢仕上げの場合，周りの景色がプリント表面に反射して映り込んでしまうことがあるので，デジカメで撮ったものをよく確認すること。フラッシュを使うと思いっきり映り込むので原則として発光禁止にする（図1）。デジカメならその場で確認してなんぼでも撮り直しがきくので，こういう用途（要するに複写）では銀塩写真に比べて有利といえるだろう。

映り込みを防ぐには，まず，光源を真っ正面に置かないこと。光源を真っ正面（つまり，カメラの真後ろ）から当てると，光源自身が映り込んでしまう。フラッシュを発光させないほうがいいのも同じ理由である。また，光源を自分の背後に置くと，自分自身の影にも悩まされることになる。

次に，カメラは黒っぽいほうが有利である。黒くても白くても映り込んでいることには変わりないのだが，白いほうが断然目立つのである。もし，カメラが白っぽい場合は，黒い布で覆うなどするとよいだろう。

同じような理由で，カメラの後ろも黒いほうがよい。暗幕などがある場合にはそれが背後にくるようにセッティングすればよい。そういうものがなくても，習字の下敷きを使うとか，自分が黒っぽい服を着るとか，黒い手袋をするとか，各自でいろいろ工夫してほしい。まあ，通常そこまでやる必要はないと思うが，スキャンする写真や部屋の条件などでどうしても映り込みが消えない場合はやってみる価値はあるだろう。

あと，意外に目立つのがホコリである。スキャン前にプリントのホコリは綺麗に掃除しておこう。指紋がついているとみっともないから，身に覚えのある人は手袋などを着用したほうがいいかもしれない。

実際にプリントした写真からオリンパスC-420Lでスキャンしたものを載せておく（図2）。明るさとシャープネスは若干調整してある。C-420Lは画素数が640×480ピクセルで，ズームがなく，マクロモードにしても最短撮影距離が20cm前後と，こういう用途にはかなり厳しいものがあるが，それでも電子メールで送るとか，Webに掲載するとかの用途には十分使えることがわかるだろう。

中級編
リバーサルフィルムのスキャン

リバーサルフィルムで撮影するような人はライ

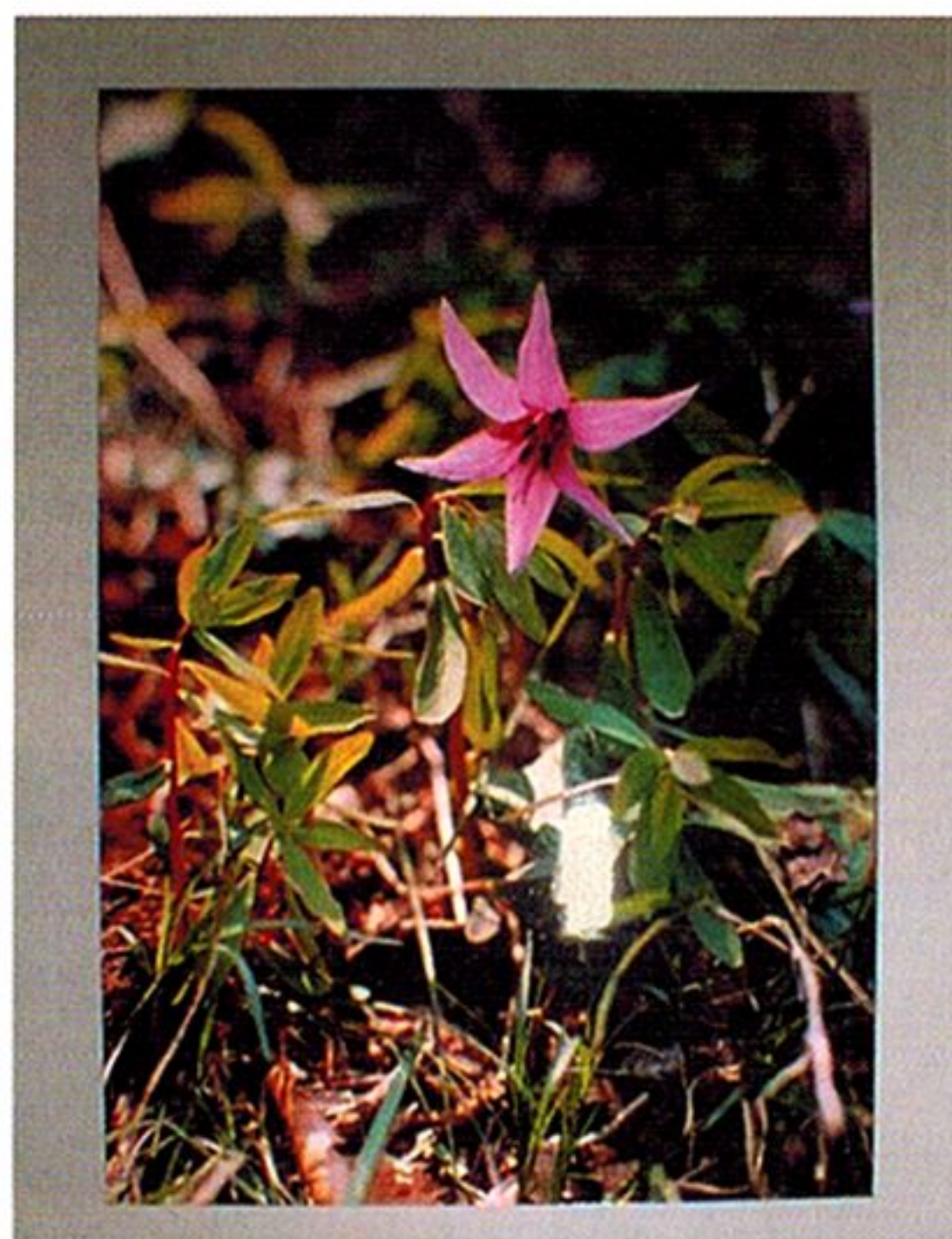



図1　フラッシュを発光させるとこうなる。プリントの中央よりやや右下の白い長方形が映り込んだフラッシュ。はっきりと映り込んでいるうえに，撮影距離が近いために場所によってフラッシュからの距離が大きく異なってしまい，明るさにムラが生じている

図2　実際にプリントからスキャン（？）したもの

デジカメ
ルーペ
ライトビュア

図3　デジカメスキャナの図

目で見た場合　デジカメで撮影した場合
網膜　CCD
水晶体　デジカメのレンズ
ルーペ
フィルム

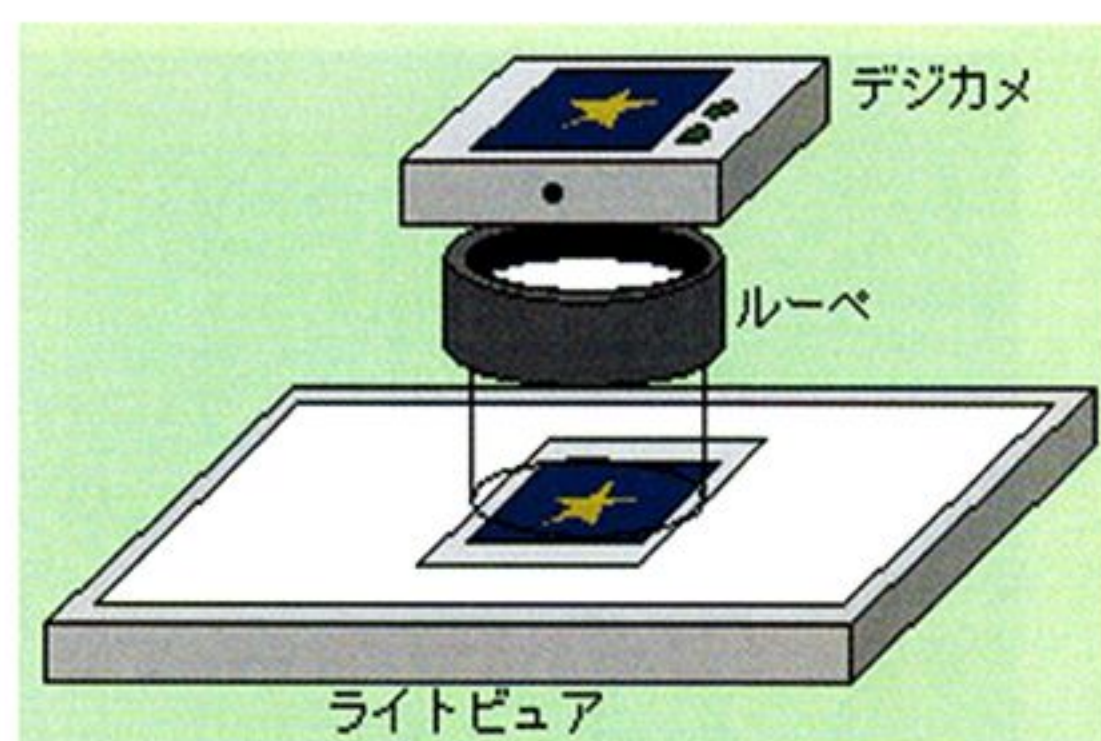
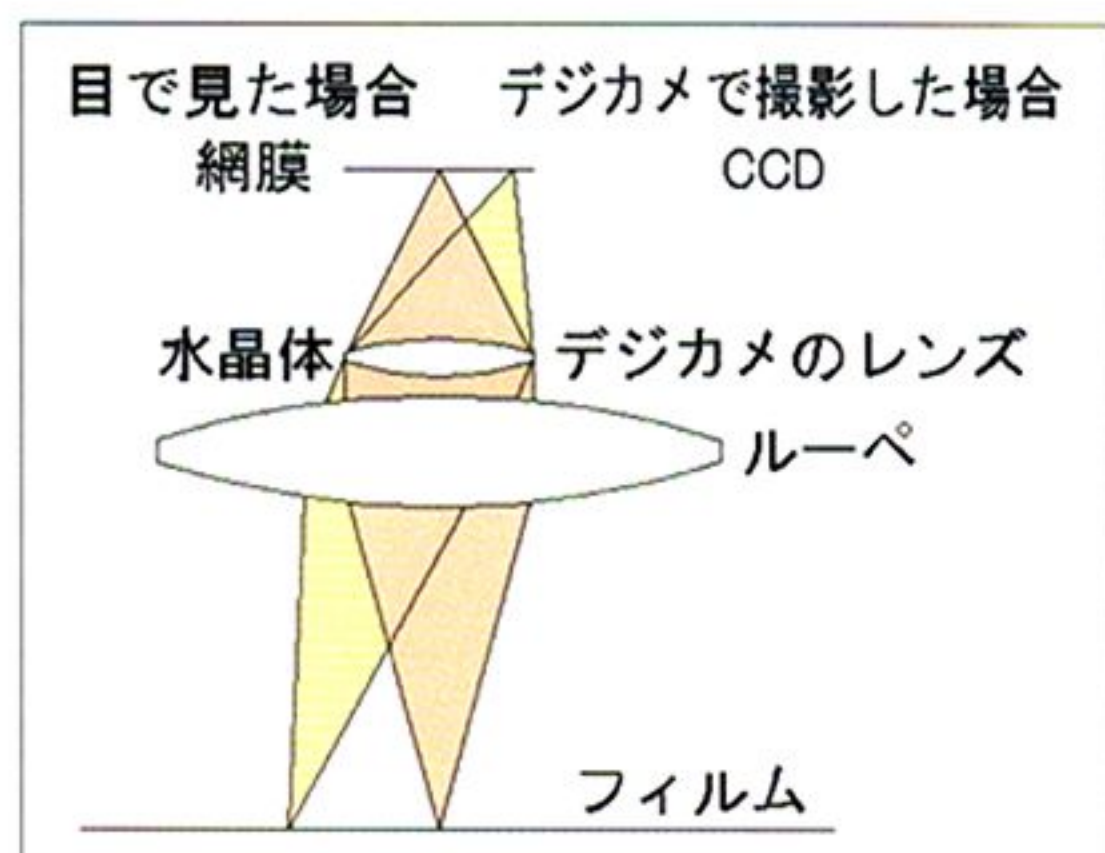



図4　デジカメスキャナの原理

トビュアとルーペを持っていると思うが，これを使ってスキャンを行う。デジカメがマクロ（接写）撮影できるようなら，フィルムをビュアの上に載せ，デジカメで撮るだけでなんとかなる場合もあるだろう。

なんとかならない場合は，フィルムの上にルーペを載せ，その上から撮る。つまり，普通に目でルーペをのぞいている状態にして，目の代わりにデジカメを置くというイメージである（図3，4）。天体撮影や顕微鏡撮影で「コリメート法」という撮影方法があるが，原理的にはこれと一緒である。このとき，視度調整ができるルーペを使っていて，なおかつデジカメのマクロ機能が貧弱な場合は，要するに「デジカメが遠視だ」ということだから，ルーペの視度は遠視気味にしたほうが楽になる。

ルーペの倍率は通常よく使う4～5倍内外が使いやすい。あまり倍率が高いと像の歪曲がひどくなるので注意。また，端のほうほど歪曲がひどいのが普通だから，ルーペは大きいほうが有利である。大きいルーペは値段も高いのだが……。

光源はライトビュアがあるのでフラッシュは発光禁止。それと，条件によってはフィルムやルーペの表面に部屋中のものがこれでもかと映り込むので部屋は真っ暗にしたほうがよい（テレビの撮影と同じ理屈）。さらに，ライトビュアの明かりがデジカメ本体を照らし，デジカメ本体がフィルムに映り込んでしまうこともあるので，そういう場合はライトビュアの光が直接見えている部分を黒い紙やアルミ箔などで覆い，それでもダメな場合はデジカメを黒い紙で覆うとよい。もちろん，レンズの部分には穴を開ける。じゃないとスキャ

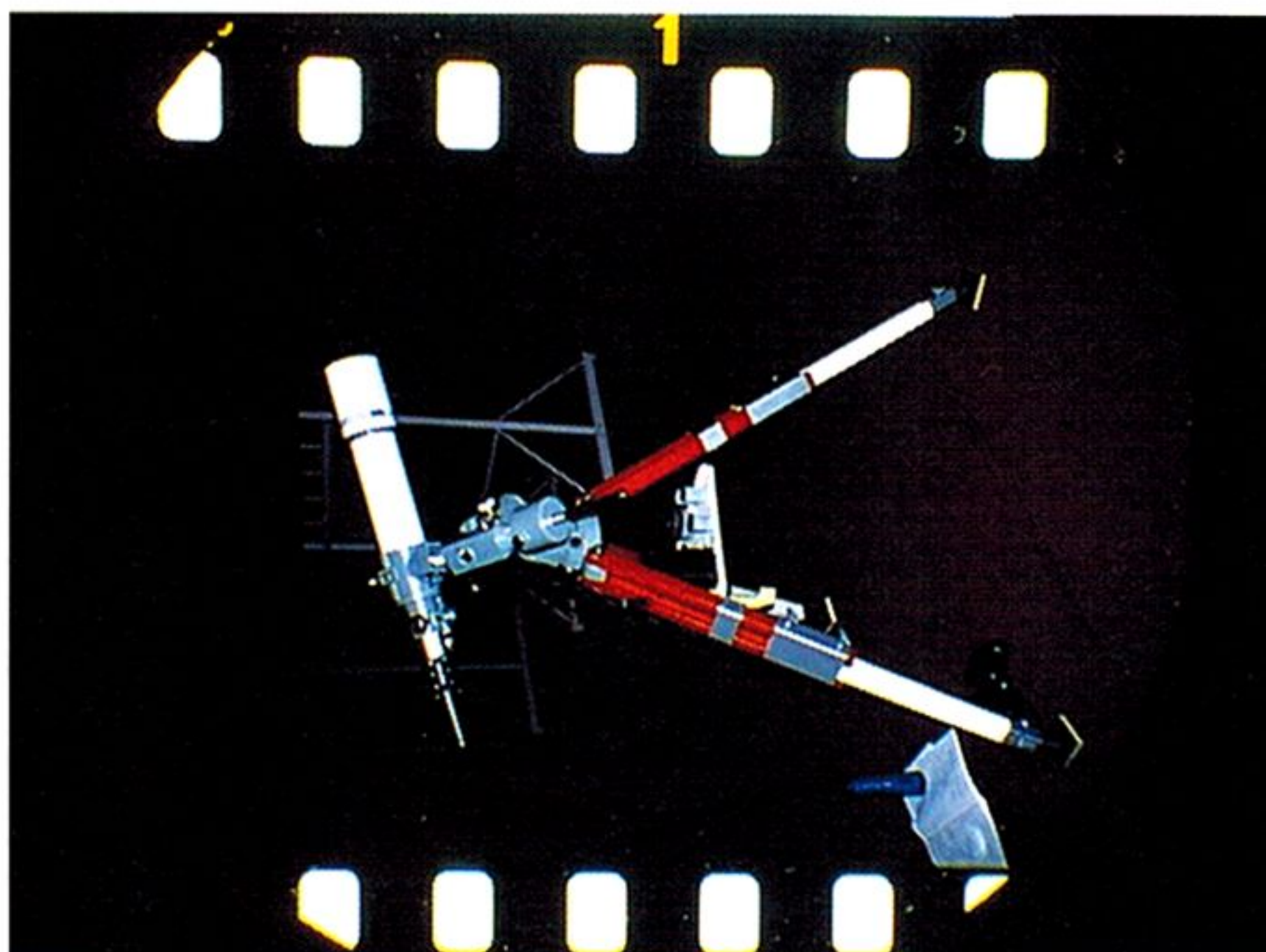



図5　リバーサルからのスキャン結果。歪曲の度合と倍率，両方とも満足できるルーペが手元になかったため，50mmのレンズをルーペの代わりに使っている。上級編参照

図6　段階露出したネガ。左からアンダー・標準・オーバー

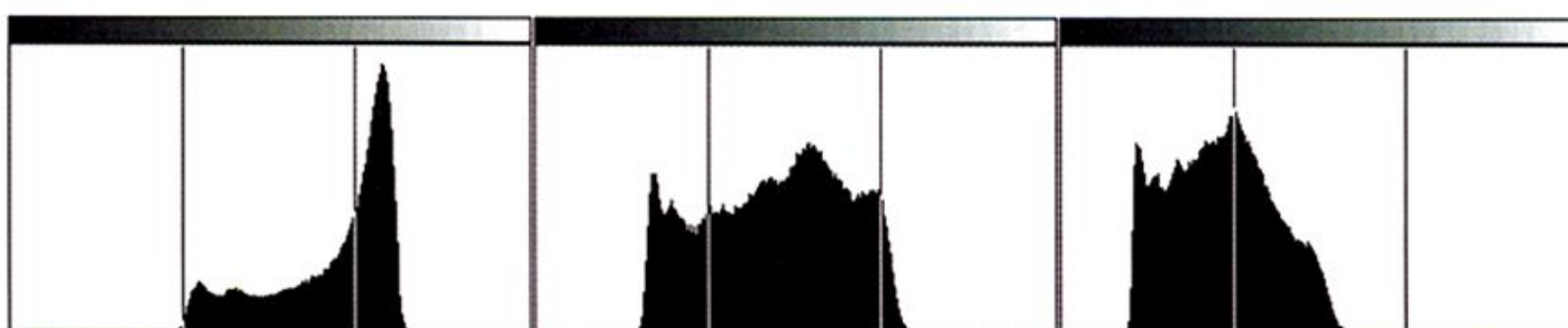



図7　ヒストグラム

図8　できあがったプリント

ンできない。

うちのデジカメはオリンパスのC-420Lなのだが，こいつはごていねいにレンズの周りがツルテカの銀色になっていて，なにをやってもこの部分が映り込んでしまうため，結局，レンズの大きさにくり抜いた黒い紙で覆ってスキャンした。そうやって苦労して撮影したものが図2である。

上級編

ネガフィルムのスキャン

この調子でネガもスキャンできる。ただ，普通のデジカメにはネガ/ポジ反転機能はないから，画像処理ソフト（いわゆるフォトレタッチソフト）でネガ/ポジ反転を行うことになる。もし，プリントがあれば見比べながら色調調整ができるが，「プリントがないからネガからスキャンしてるんでい」という人は，根性で色調を調整するしかない。

デジカメでのスキャン（？）自体はリバーサルの場合と一緒なのでよいだろう。リバーサルで写真を撮らない人はルーペやライトビュアなどは持っていないと思うが，ルーペは虫眼鏡や一眼レフのレンズ（50mm前後の明るいレンズがよい）などで，ライトビュアは曇りガラスにフィルムを貼りつけて外光で代用するなど，手はいろいろあるので工夫してほしい。一眼レフのレンズをルーペの代わりに使う場合，絞り連動ツメを開放の位置に動かし，綿棒などを詰め込んで固定（極悪……）して使う。

スキャンするとき，画像の大きさはできあがりの2倍程度にしておいて，あとで縮小したほうがよい結果になる。ネガをデジカメで撮影してヒストグラムを見るとわかるが，ネガはヒストグラムの全域を使っていないのである。「ネガはラチチュードが広い」，つまり，露出をちょっとくらい失敗しても見れる写真になるのだが，その理由がここにある。

図6の3枚の写真は，絞りは変えずに，シャッター速度をカメラの指示速度と，4倍（2段）短い速度（露出アンダー），4倍長い速度（露出オーバー）の3通りに変えて撮影したものである。露出アンダーになるとネガは色が薄くなり，したがってヒストグラムは明るいほうに偏る。オーバーはその逆である。

が，同時プリントしたもの（図8）を見てみると，露出アンダーのネガからプリントしたものはちょっと苦しい出来だが，指示速度どおりのネガと露出オーバーのネガからプリントしたものはほとんど同じ仕上がりになっている。つまり，印画紙に焼く段階で，ヒストグラムの移動と引き伸ばしが行われていることになる。

逆にいうと，デジカメでネガをスキャンした場合，きちんとした画像にするためには，画像の濃淡を反転させるだけではなく，ヒストグラムの移動・引き伸ばしの処理も行わなければならない。引き伸ばすとヒストグラムは櫛状になり，画像が

図9　片栗の花（ネガ）

図10　反転したところ

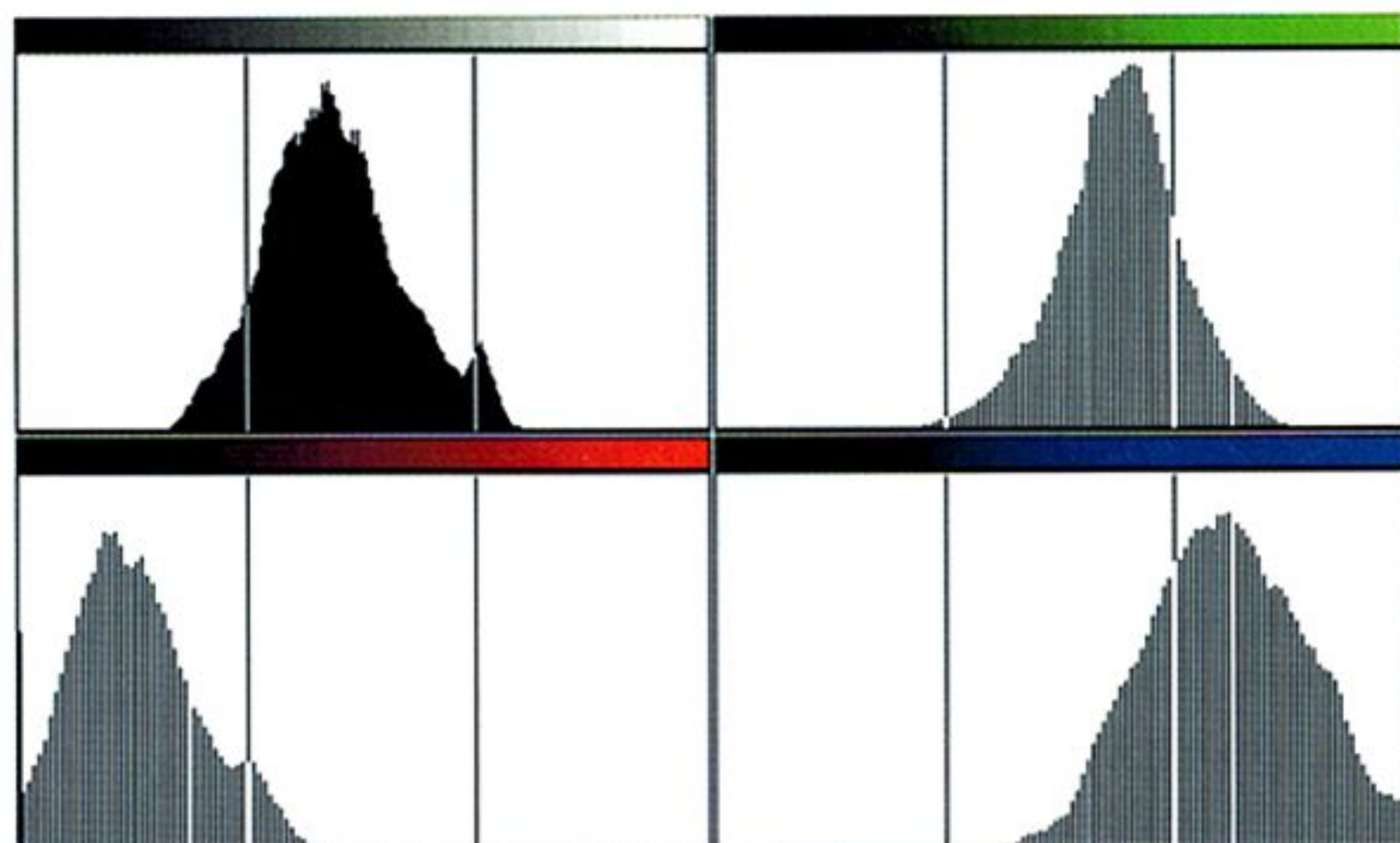



図11　ヒストグラム

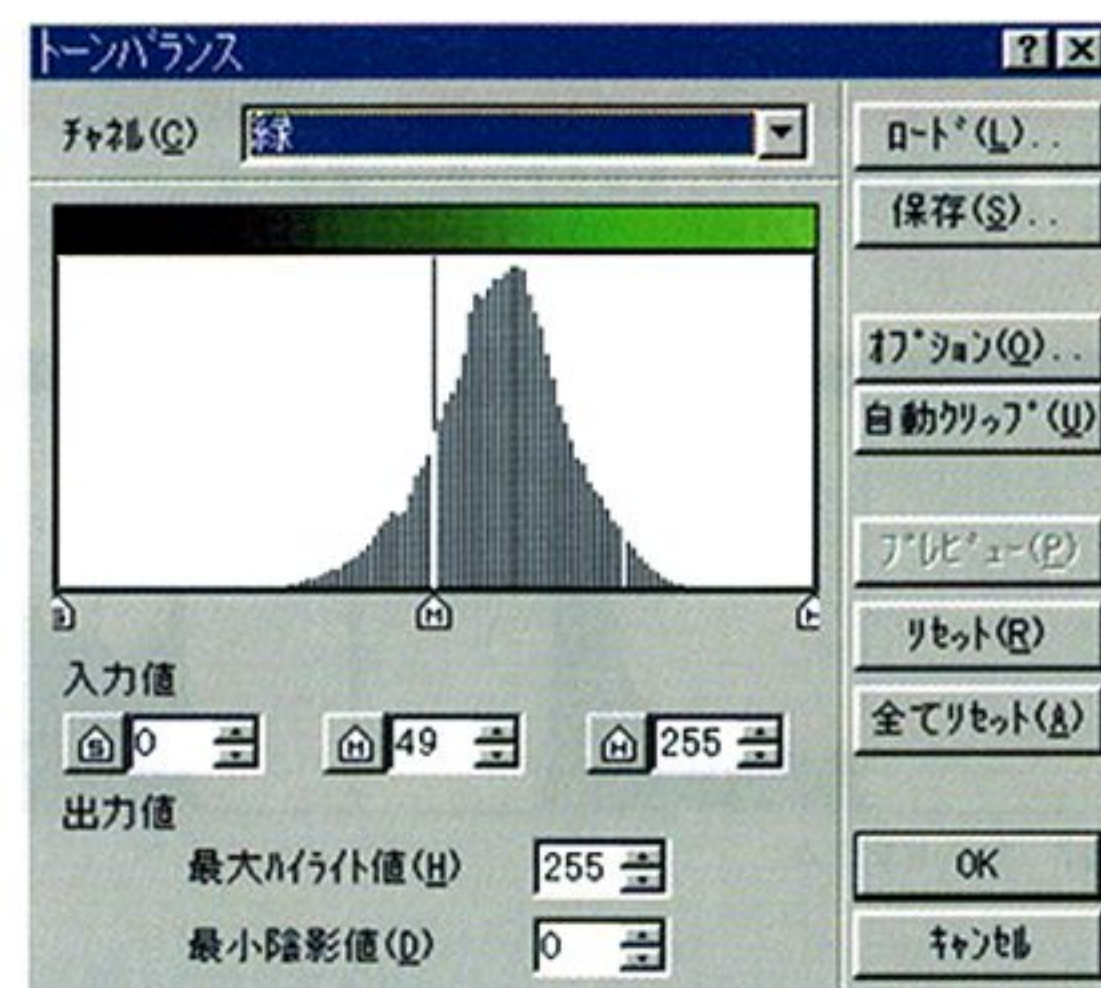



図12　トーンバランス（緑・操作前）

図14　トーンバランスの操作を終えたところ

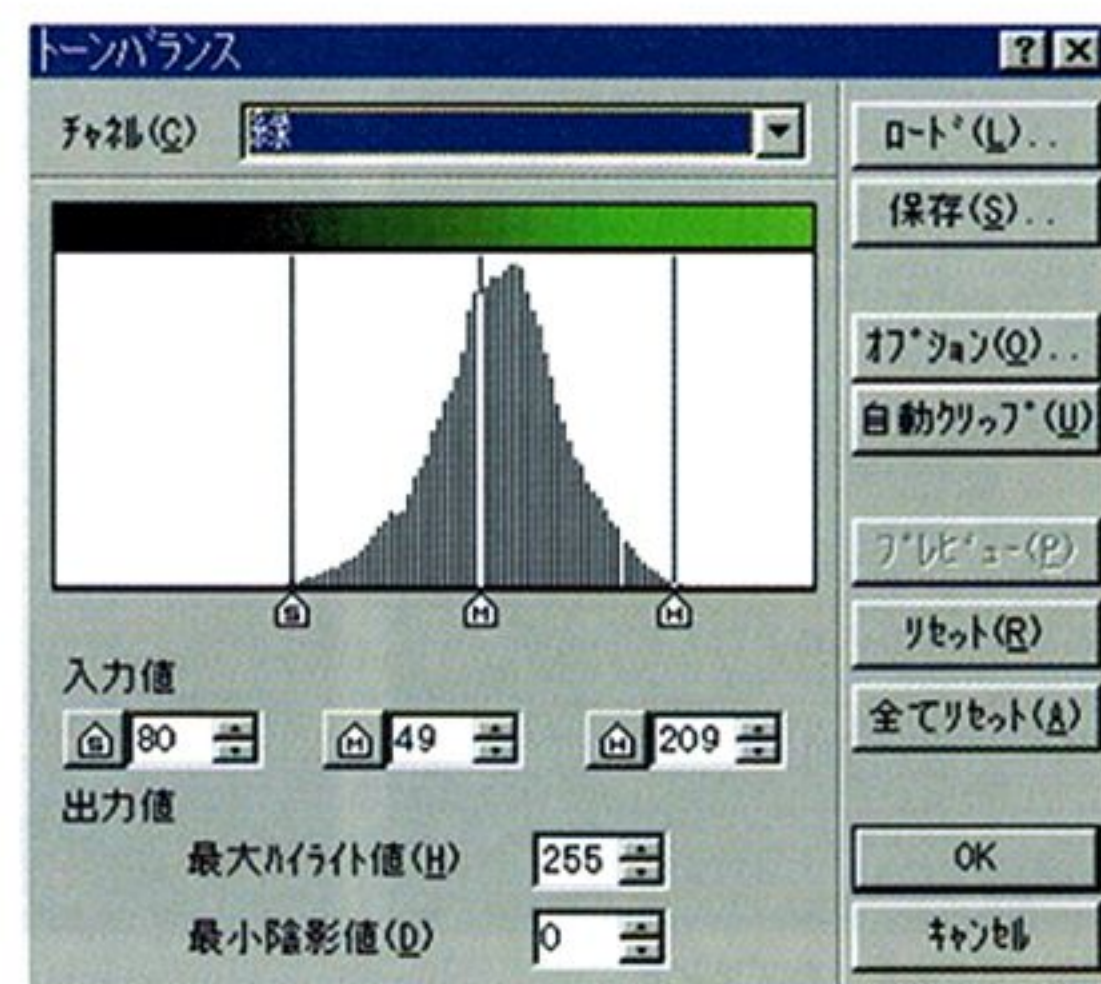



図13　トーンバランス（緑・操作後）。「S」「H」の位置をヒストグラムの裾野よりちょっと中央寄りに持ってくる。RGB各色について行う

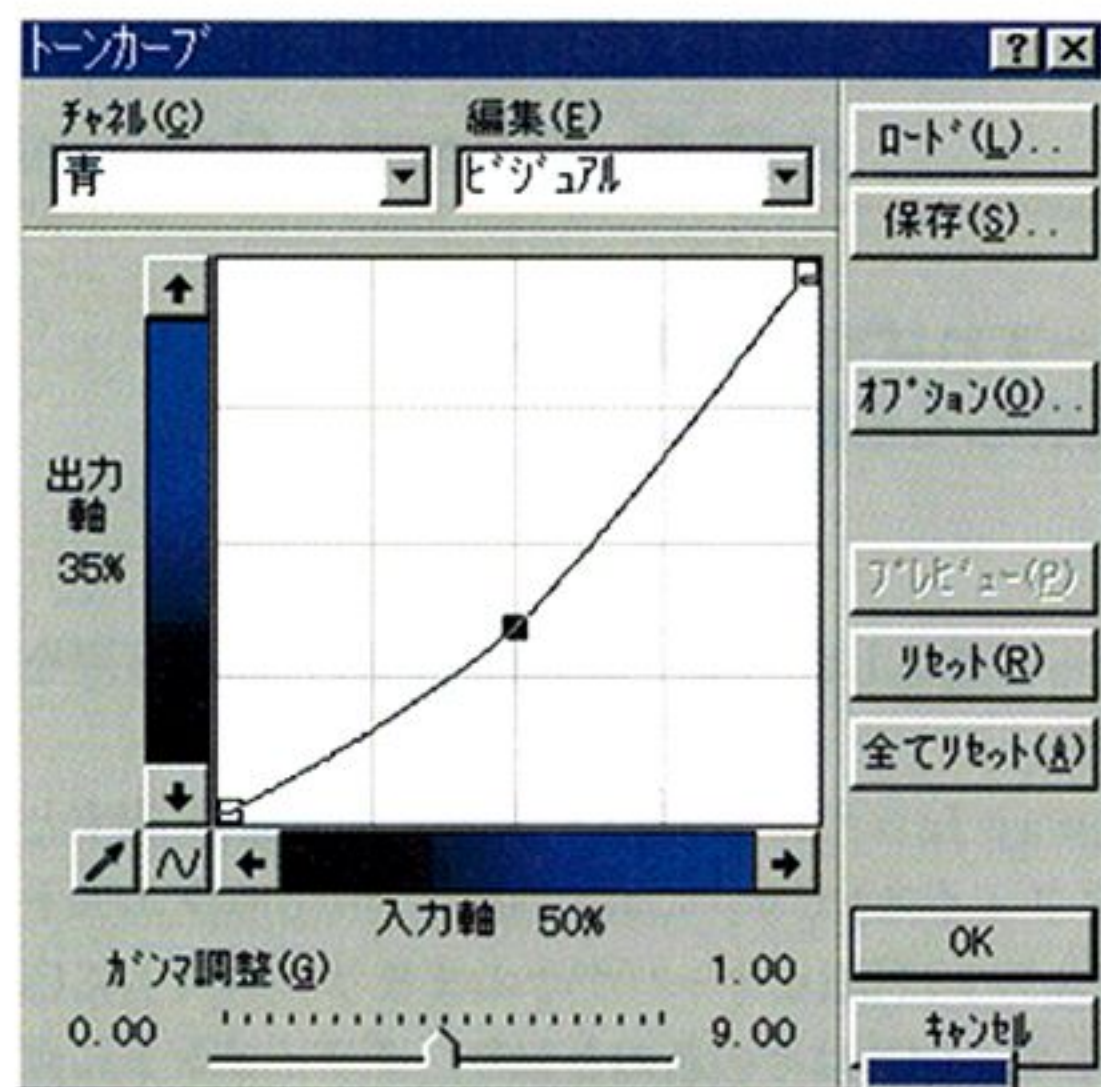



図15 トーンカーブの調整（青）。あんまり凝りすぎるとかえって変になるので，この程度にとどめておいたほうがよい

図16 色調を調整したところ

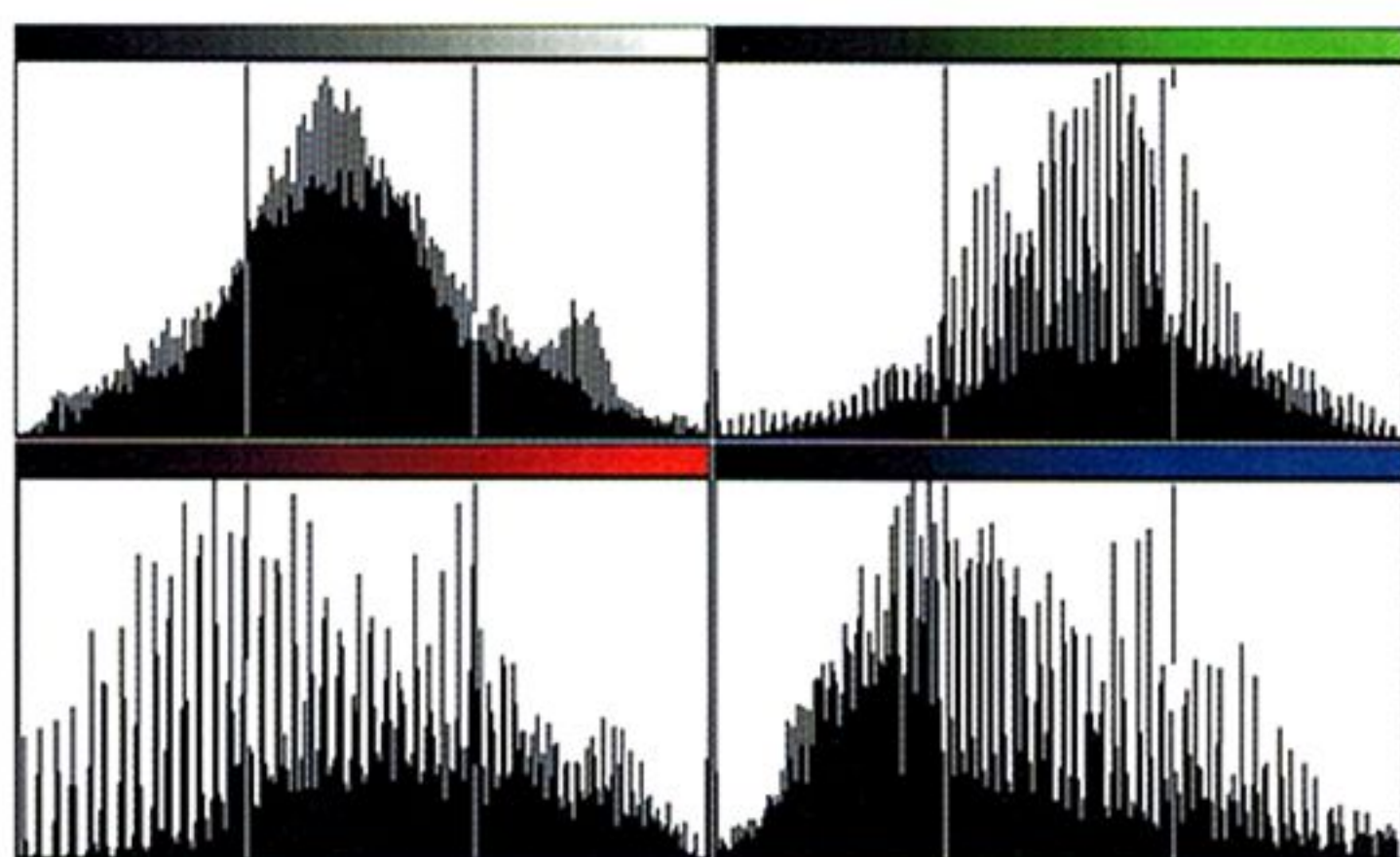



図17 ヒストグラム。元のヒストグラムと比べ，櫛の歯と歯の間が広がっている

図18 できあがり

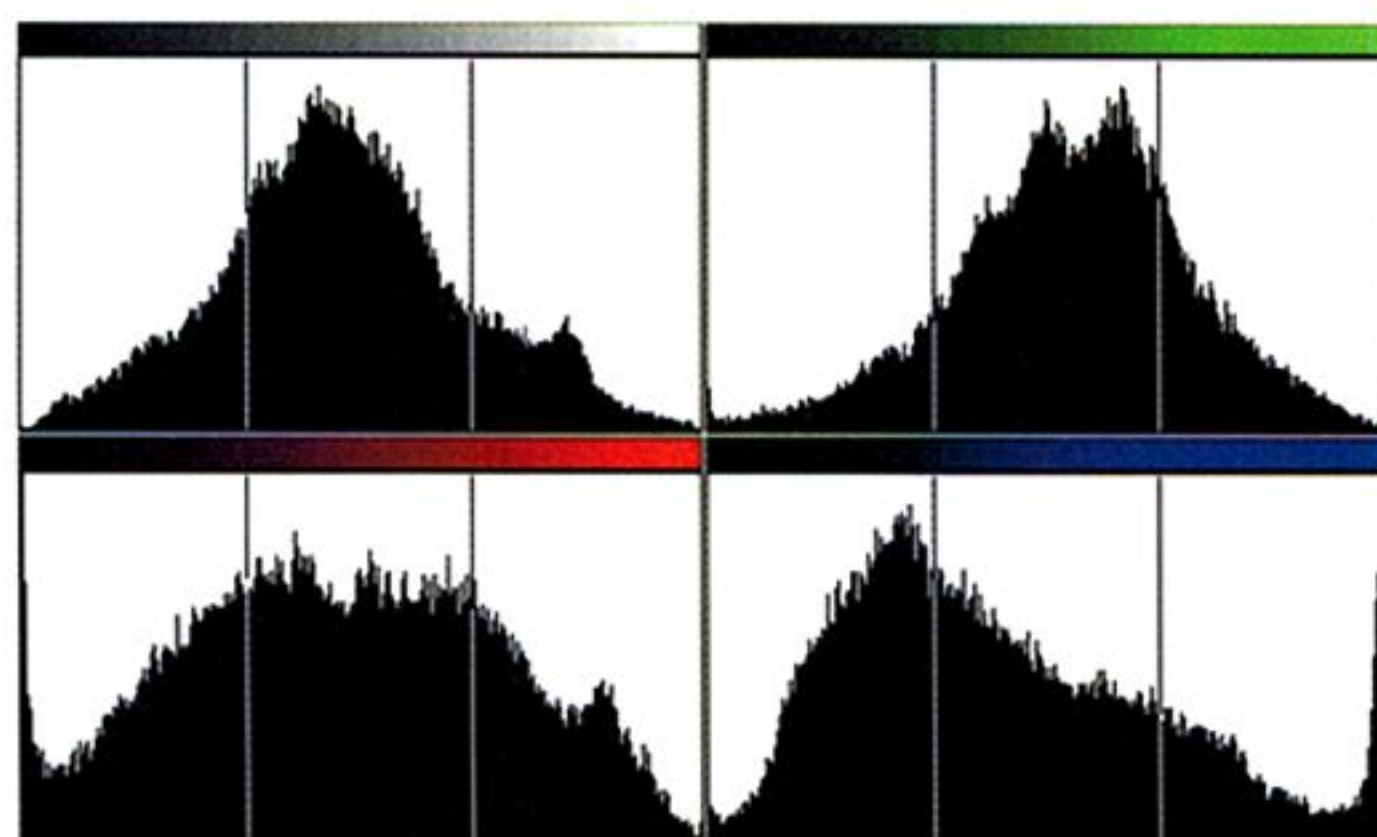



図19 ヒストグラム

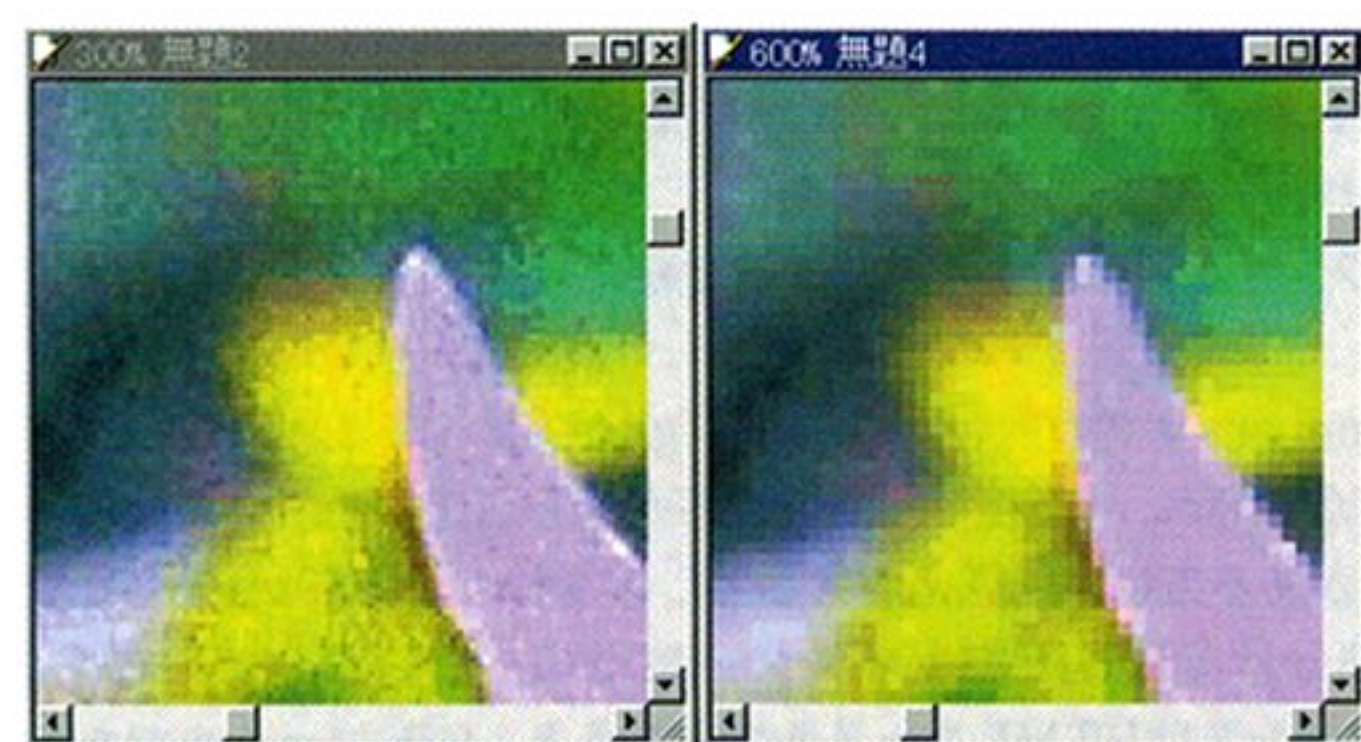



図20 縮小前と縮小後を同じ大きさに見えるように拡大して並べたもの。左が縮小前，右が縮小後。縮小後のほうがモザイク模様は粗くなるが，全体的に滑らかに見える

図21 COOLSCAN Ⅲでネガから取り込んだもの

ザラザラした感じになってしまう。ところが，これを縮小すると，隣接画素の平均が取られることになってヒストグラムは綺麗な形に戻り，画質も滑らかになる。

まあ，高〜いフォトレタッチソフトを使っている人には関係ないことかもしれないが，いまあるソフトを生かそうという向きには，大き目の画像を用意するか，一度拡大して色調を調節してから縮小したほうがよい結果が得られる。で，実際にネガをC-420Lで撮影してみたのがこれ。

撮影したらフォトレタッチソフトへ突っ込んで，ネガ/ポジ反転する。MicrografxのPhoto Magicでいうと，「イメージ」「反転」である。今回の場合，画像が回転してしまっているので，ついでにここで直しておく。これでなんとなく見れる画像にはなったと思うが，ネガのフィルムベースは赤っぽいことが多いので，反転すると妙に青っぽくなる。

ヒストグラムを見ると**図11**のようになっていて，赤は暗いほうに，青は明るいほうに偏っている。RGBのヒストグラムが櫛状になっているのは，元がJPEG画像だったから，のようだ。とりあえず，色の偏りを直すことにする。Photo Magic 4だと「補正」「トーンバランス」を使うとヒストグラムを見ながら調節できるので楽だが，Photo Magic 6ではヒストグラムが一部入れ替わっているようなので注意（私の持っているバージョンだと，赤と青が入れ替わっている）。お気楽に済ませたい向きには自動クリップを使ってもいいだろう。

これで普通に見られる画像になっただろう。まだちょっと青みが強く感じられるので，青を弱く，赤を強く調節してみる。ついでに，色のノリが悪いようなので，彩度も上げてみよう。これで遠目には綺麗な写真になるが，ヒストグラムを見るとものの見事にボロボロになっている。

ここで画像を1/2に縮小する。すると，画像が小さくなったのと引き換えにザラザラした感じがなくなる。ヒストグラムを見ると，櫛の谷間が埋まっていることが分かるだろう。縮小したせいで少しぼやっとした感じになってしまったので，最後の仕上げとして軽くエッジを強調しておく。Photo Magicだと「イメージ」「エフェクト」の中に「シャープ」というのがあるので，これで強いエッジだけをちょっと強調すればよいだろう。アンシャープマスクなども結構使える。これでできあがり。参考までにあとからネガをCOOLSCAN Ⅲでスキャンしたものも掲載しておく。ちょっと色を濃くしすぎたかな。

ここまで力説しておいてなんだが，労力の割に報われないという話もあるので，どーしてもプリントが見つからないのだけど画像はいますぐほしいときなど，最後の手段として頭の片隅にでも入れておく程度でよいだろう。ちなみに，サンプル画像に登場した花は片栗の花である。周りの葉っぱは片栗の葉っぱではないのでご注意を。片栗の葉っぱは画面中央下，わらくずみたいなのの奥に，向こうから手前に向かって生えている，比較的大きな竹の皮模様の葉っぱである。

Webページでグラフを表示しようII

大和 哲　Yamato Satoshi

今度は本当にその場でGIF画像を生成していくことにする。この手法をマスターすれば非常に多彩なページが作れるようになるだろう。外部ライブラリの呼び出し方と使い方についても詳しく解説する。

97ページのパートIのほうでは，CGIにContent-type:text/htmlでHTMLを出力して，＜IMG＞タグのwidthオプションでgifイメージファイルを引き伸ばして棒グラフに見せたわけですが，ほかの形式，たとえば折れ線グラフや円グラフを作りたいときにはどうしましょう？　先ほどのグラフィックを引き伸ばす方法でなにができるでしょう。制限時間5秒……いや，考え込むまでもなく，棒をまっすぐ引き伸ばすんだから棒グラフしかできませんよねぇ，それは(棒の中で割合を示す，割合棒グラフくらいなら作れるかもしれませんけど……それもやっぱり棒グラフ)。でも，いまどき携帯電話でも絵が見えるんですもんね，やっぱりちゃんと画像を出力して好きなグラフでもなんでも表示させてみたいもんです。

Web上を探してもあまり例がないのが不思議ですが，実はCGIを使って好きなグラフィックを出す方法は存在します。少なくともこのページのサンプルプログラムはしばらくのあいだ私の日記ページで毎日アクセス数を折れ線グラフで表示していましたから。このページでは，CGIでグラフィックを出力して，折れ線グラフ式のアクセスカウンタを作ってみましょう。

このページのサンプルプログラムは以下のようなことが許されているwebサーバ環境で動作します。

・telnetでシェルアクセスできる
・Perl5が使える
・cgiが動く
・Cでプログラムをコンパイルできる

それから，当然ですが，実行結果を見るにはグラフィックを表示できるブラウザが必要になりますね。

なぜCGIでグラフィックが出力できるのか

CGIプログラムの使い方に入る前に，まずはこのプログラムの動作原理をご紹介しましょう。ちょっと見逃しがちな，基本的なCGIの動作原理をこれで思い出してくださいね。

CGIとはCommon Gateway Interfaceの略で，簡単に日本語に訳すと「共通にデータが通るインタフェイス」です。これが通常のhttpリクエストと異なるのは，通常のリクエストはhttpサーバがそのままリクエストに応答してデータを返すのに対して，CGIはその名のとおり，クライアントに直接データを返さず，プログラムがデータゲートウエイとなってなんらかのデータを加工して結果を返す，ということなのです。

さて，httpリクエストに対してwebサーバは指定されたファイルがhtmlだったらレスポンスヘッダとHTMLで書かれたコンテンツをインターネット上に流します。そしてクライアント(ブラウザ)がそれを表示するわけですね。テキスト(HTML)を表示する普通のCGIではサーバ側からレスポンスヘッダ，コンテンツの順にクライアントにデータを発信するわけですが，cgiプログラムは，

```
print "Content-Type: text/html¥n¥n";
```

のようにContent-Type：ヘッダと改行を2回送り，それからHTMLに準拠した文法で書かれたコンテンツを標準出力に送っていました。

ここで問題なのですが，もしhttpリクエストを，

```
http://www.foo.com/bar.gif
```

のようにgifファイルをリクエストした場合，サーバからはどのようにデータがブラウザに流れてくるのでしょうか？

さて，このhttpのレスポンスですが，GIFイメージファイルのリクエストについてはどのように返されるのでしょうか。試してみましょう。ここでは，apacheのサンプルページに添付される「apache_pg.gif」というファイルを参照してみました。telnetでwebサーバの80番ポートに接続し，GET文でgifファイルのファイル名を指定します。

```
GET /apache_pb.gif HTTP/1.0
```

すると，次のようにレスポンスは返ってきます。

```
HTTP/1.1 200 OK
Date: Sun, 23 May 1999 05:33:41 GMT
Server: Apache/1.3.3  (Unix)
Last-Modified: Wed, 03 Jul 1996 06:18:15 GMT
ETag: "14aea-916-31da10a7"
Accept-Ranges: bytes
Content-Length: 2326
Connection: close
Content-Type: image/gif

GIF89a　N離ウｧｒ sskkkZZZ!述洵　;B領峠銃1J間{渣ノ憮i慶
憮qc!ガ GH
(以下GIFファイル内のバイナリの内容が続く)
```

Content-Type：ヘッダの内容が「image/gif」になった以外は同じようにヘッダの内容が送られ，1行空けてその内容が送られます。ダンプリストなどで見たことがある方には見覚えがあるでしょうが，「GIF89a……」と表示されている部分は実はGIFファイルの内容そのものになっています。HTTPでのレスポンスはHTMLファイルであろうと，GIFファイルであろうと，同じようにHTTPレスポンスヘッダが送られ，コンテンツ内容そのものがそのあとに送られるというかたちをとり，そのあとにコンテンツ内容

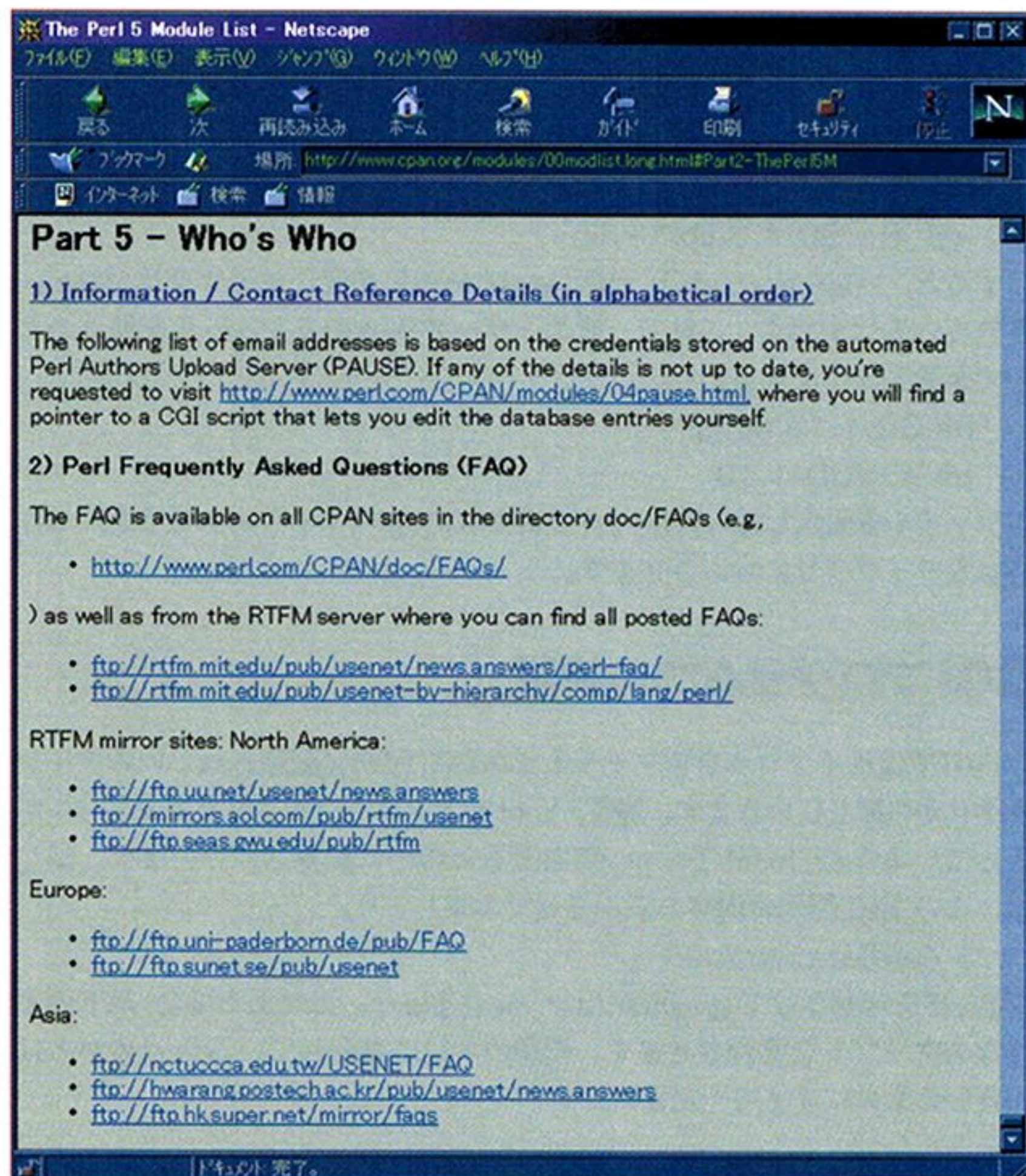



図1　Perlのモジュールを探すならここ

図2　GDモジュールの最新版はここにもある

そのものがくる，ということで基本的に変わりはないわけです。

画像を動的に生成するには

さてさて。本題ですが，CGIプログラムでグラフィックイメージをサーバから出力するにはどうしたらいいでしょう。いままでの結果から想像できるとおりです。そう，

```
print "Content-Type: text/html¥n¥n";
```

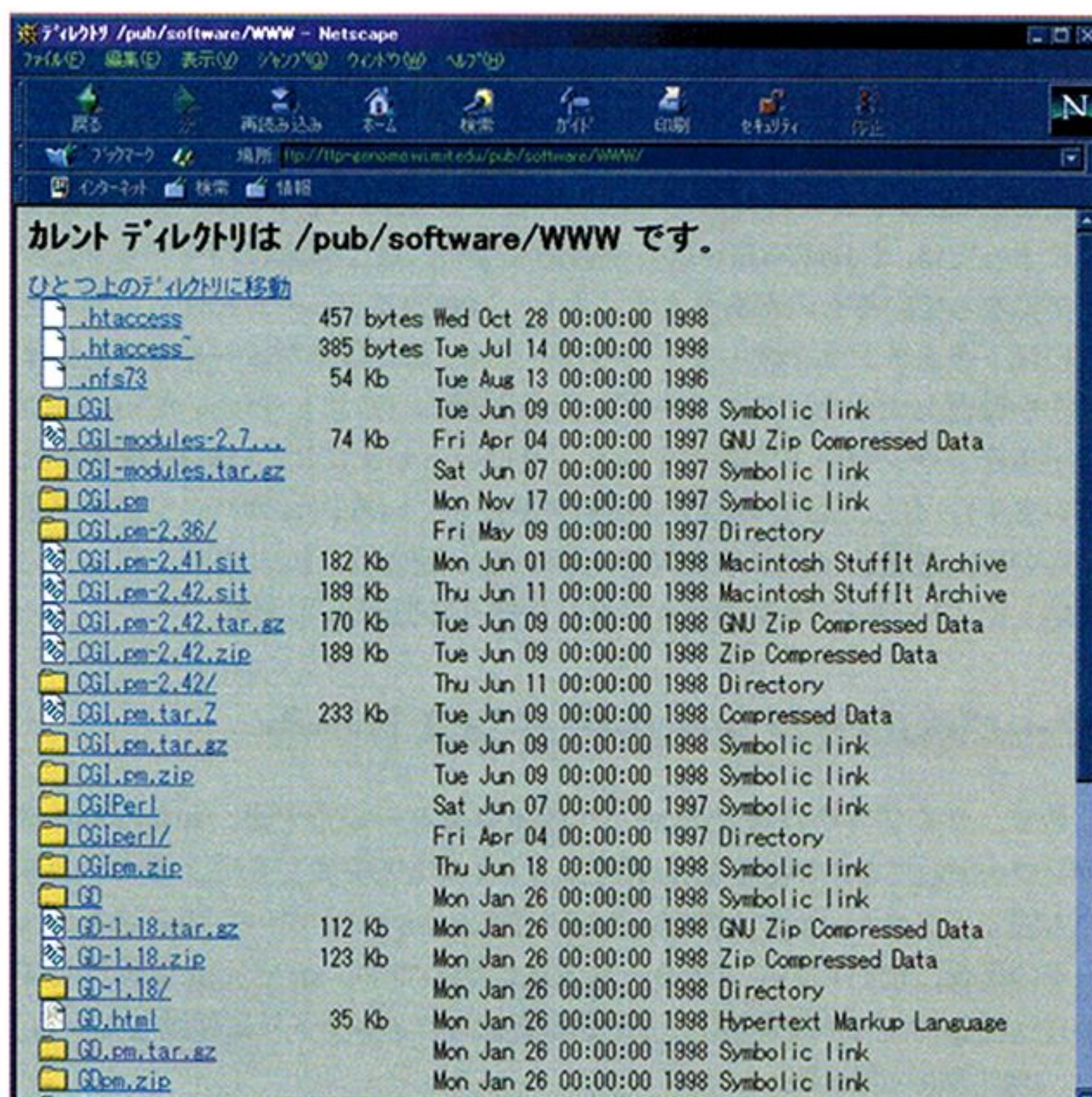
を出力してから，GIFイメージをネットワーク上に流せばいいのです。この辺の話は復刊1号でも書いたようにW3Cによって決められていて，RFC 1945で定義されています。これによると，イメージにせよ音楽ファイルにせよすべてのファイルは(ブラウザさえ対応していれば)コンテントタイプを出力してそのあとコンテンツを流せばいいのですね。RFC1945についてはW3CのWebサイトの，

http://www.w3.org/Protocols/rfc1945/rfc1945/

を参照してください。

ただ，問題がひとつあります。それはすでにできているグラフィックイメージをただ流すのは問題ないのですが，グラフィックを作り出したり，加工したりするのは，生のPerlだけでは結構面倒です。そのような関数は用意されていませんからね。そのために，Perl5にはいくつかグラフィックイメージを生成したり加工したりするためのライブラリ類が配布されています。そのなかでも有名なのがImageMagikやGDモジュールです。

今回のサンプルプログラムではグラフィックイメージを生成するのにGDモジュールを使います。このGDモジュールはもともとは，コールドスプリングハーバ研究所のQuest Protein Database Centerで開発されたGIF作成のためのCライブラリで，このGD.pmは主にGIFイメージを生成したり操作するためにPerl5用に移植されたダイナミックロードモジュールです。このモジュールでは主に，

直線，ポリゴン，四角形，弧の描画
ブラシやタイル描画
破線を引く
文字列描画
透明色とインターレースGIF設定

といったことができます。

このGDモジュールをダウンロードしてきて，CGIプログラムから使って，できたGIF画像をネットワーク上に(CGIプログラムから見れば単なる標準出力ですが)流せば，見事にブラウザ上に，その場でリアルタイムに生成したグラフィックを表示することができる，というわけです。

Perl5モジュールはどこにあるの!?

では，このGDモジュールを入手しましょう。GDモジュールはこのCPANからダウンロードすることができます。この原稿を執筆している現在，いちばん新しいGDライブラリはGD-1.19のようでモジュールはここにありました。

http://www.cpan.org/modules/by-category/18_Images_Pixmaps_Bitmaps/GD/GD-1.19.tar.gz

GDモジュールに限らず，perl用のモジュールで「こんな機能のものがあったらいいのになぁ」と思ったら，CPANから探すのがいちばん見つけることのできる確率が高いでしょう。CPANとは「包括的Perlアーカイブネットワーク」の略で，その名のとおり，PerlそのものやPerl用のソフトウエアや特にライブラリなどを包括的にまとめているサイトです。ホームページはここになります。

http://www.cpan.org/

このサイトはいくつかミラーサイトがあり，マルチプレックス機能で，ネットワーク的に一番近いサイトを自動的に接続してくれますので，サーバが極端に重くなることもそれほどありません(ただし，ドキュメントはすべて英語になっています)。Perlのモジュールを探す場合，カテゴリ別のモジュールの一覧が，

http://www.cpan.org/modules/00modlist.long.html#Part2-ThePerl5M

にありますので，ある程度の機能(グラフィック関係とか，データベース関係とか…)や名前がわかっているときはここから探すと楽です。

ついでですが，(ドキュメントを見てみないとわからないわけですが)，こ

のGDモジュールに関してはこちらにも最新版があるようです。

ftp://ftp-genome.wi.mit.edu/pub/software/WWW/GD.pm.tar.gzGD

なお，GDモジュールはそうではないようですが，Perlのモジュールはモノによっては，Perlの5.00xにしか対応しない，など非常にバージョンにシビアになっているものがあります。もし，ほかのモジュールを使ってみるときにはドキュメントを読んで，どのバージョンのGDモジュールを使えばよいかを確認してからダウンロードしたほうがいいでしょう(たいてい，モジュールをダウンロードするディレクトリにいっしょにREADMEも置かれています)。もし，お使いのプロバイダが使っているPerlのバージョンが古いもので，最新のイブラリが対応していない場合は，Perlのバージョンに対応した古いバージョンのライブラリを使う必要があります。

サーバへのモジュールのインストール

さて，ライブラリのWebサーバへのインストールですが，もちろん，ダウンロードしてきたINSTALLドキュメントなどを読んでから，そのとおりの方法でインストールするのがいちばん正しいわけですが……Perlのモジュールの場合，たいていインストール方法は同じです。suでrootに移行してからtar.zipファイルを解凍し，cdで解凍先のディレクトリに移動してから,

```
perl Makefile.PL
make
```

でモジュールをお使いの環境にあわせてコンパイルして,

```
make install
```

でPerlのライブラリのディレクトリ(たとえば/usr/local/lib/perl5など)にファイルをコピーします。GDモジュールの場合も，同じで，自分がroot(特権ユーザー。WindowsNTではAdministrator)で自由にマシンを設定できるのであれば，単にライブラリをインストールすれば問題なく使うことができてしまいます。

ただし，この方法はプロバイダにホームページの領域を借りてCGIを実行しているなど，一般ユーザーでサーバにloginしている場合は使うことができません。なぜなら，PerlのモジュールはPerl5のデフォルトのライブラリディレクトリにインストールします。ライブラリの収納されているディレクトリはたいていの場合，セキュリティ上の問題で，読み込みと実行は一般ユーザーでもできますが，書き込みのほうはrootでしかできないようになっているからです。通常一般ユーザーは，ユーザーの与えられたホームディレクトリ以下しか書き込みすることはできません。そこで，一般ユーザー権限でほかからダウンロードしてきたPerlのモジュールを使う場合には，そのモジュールを自分のディレクトリにインストールしておき，Perlのスクリプトにはそのモジュールがある場所を明示的に使うことになります。

では，GDモジュールを自分のホームディレクトリの下にインストールしてみましょう。Webサーバにtelnetでログインしてみてください。ここでは，WebサーバはUNIX系のOSを採用していて，Cコンパイラとファイル圧縮展開ソフトとしてGNU tarがインストールされているものとします。

まず，GDモジュールのアーカイブをホームディレクトリに展開します(以下，ホームディレクトリは~で表します)。GDモジュールのアーカイブファイルがGD-1.19.tar.gzだった場合,

```
tar zxvf GD-1.19.tar.gz
```

とすると，~/GD-1.19ディレクトリにファイルが展開されます。ここで,

```
cd ~/GD-1.19
```

でGDライブラリのソースが展開されているディレクトリへ移動し，通常(つまりrootでの)インストールと同じように,

```
perl Makefile.PL
make
```

でモジュールをお使いの環境にあわせてコンパイルします。これで~/GD-1.19ディレクトリ内でGD.pmモジュールのコンパイルができました。GD.pmモジュールのバイナリ自体は~/GD-1.19/libに作成されています。

さて，プロバイダでの使用可能なディスクの空きは限られていますから，必要のない部分は削除しておいたほうがいいでしょう。モジュールの本体だけをコピーしておいてそれ以外のファイルは削除してしまいます。

```
cd ~
```

でホームディレクトリに移動します。それから~/libというディレクトリを作ってそこにモジュールをコピーしましょう。

```
mkdir ~/lib
cp -R ./GD-1.8/lib/* ~/lib
```

すると，~/lib/perl/libと~/lib/perl/archに必要なライブラリがコピーされているはずです。これで一般ユーザーでPerlのモジュールを使うことができるようになりました。

```
rm GD-1.19.tar.gz
rm -r ~/GD-1.19
```

でファイルを削除したら次はPerlで書かれたcgiプログラムのほうを，今作られたライブラリに対応させます。

CGIプログラム側の解説

Perlではライブラリをデフォルトで@INC配列に記録されているディレクトリから探しにいきます。通常，rootでインストールされたモジュールが入っている/usr/local/lib/perl5などもこの中に記録されています。試しに，この`INC配列の内容を表示させてみましょう。

```
> /usr/local/bin/perl
```

でPerl5を起動させて(perl5が/usr/local/bin/perl5にある場合)以下のようなスクリプトを実行させます。先頭の#行に書かれているPerlのパスはお使いのものに変えてください。

```
#/usr/local/bin/perl
foreach $i  (@INC)
{
 print "$i¥n";
}
```

すると，こんな感じで結果が表示されるはずです。これらが，Perlモジュールの記録されているディレクトリです。

```
/usr/local/lib/perl5
/usr/local/lib/perl5/5.00502
/usr/local/lib/perl5/5.00502/i386-freebsd
/usr/local/lib/perl5/site_perl/5.005
/usr/local/lib/perl5/site_perl/5.005/i386-freebsd
```

さて，GDモジュールは~/lib以下にインストールされたわけですが，CGIプログラムなどが動作する際にはこのモジュールがどのディレクトリにあるのかがプログラム自身にわかっていないといけませんね(WebサーバからCGIを実行すると，ほとんどのWebサーバで,

```
500 Internal Server Error
```

というエラーになります。telnetなどでwebサーバにloginし，シェルから実行した場合,

```
Can't locate FOO.pm in @INC  (@INC contains:
/usr/local/lib/perl5/
/usr/local/lib/perl5/5.00502/i386
-freebsd /usr/local/lib/perl5/5.00502
/usr/local/lib/perl5/site_perl/5.005/i386-
freebsd /usr/local/lib/perl5/site_perl/5.005 .)  at - line 1.
BEGIN failed--compilation aborted at - line 1.
```

という感じでエラーが表示されます。そこで，このプログラムがGDモジュールを探せるように@INC配列にGDモジュールの入ったディレクトリを明示しておきます。

これにはuse文を使います。もし，ホームディレクトリのパスが/home/deで，ライブラリの実体がlib/perl/libとlib/perl/archにある場合(ホームディレクトリ絶対パスは,

```
cd ~
pwd
```

で知ることができます)

```
use lib '/home/de/lib/perl/lib','/home/de/lib/perl/arch';
```

これで@INC配列に/home/de/lib/perl/libと/home/de/lib/perl/archが追加されます。もし，不安だったら，

```
#/usr/local/bin/perl

use lib '/home/de/lib/perl/lib','/home/de/foreach $i  (@INC)
{
print "$i¥n";
}
lib/perl/arch';
```

としてみましょう。すると，

```
/home/de/lib/perl/lib
/home/de/lib/perl/arch
/usr/local/lib/perl5
/usr/local/lib/perl5/5.00502
/usr/local/lib/perl5/5.00502/i386-freebsd
/usr/local/lib/perl5/site_perl/5.005
/usr/local/lib/perl5/site_perl/5.005/i386-freebsd
```

という具合に，モジュールを検索するパスが増えたことがわかるはずです。ここまでできれば，あとはここから普通にPerl5でモジュールを使う要領で，

```
use GD
```

を宣言するとGDもモジュールを使うことができます。

なお，このuse文ですが，下のようにしても先ほどの例と同じように@INC配列にパスを追加できます。

```
BEGIN {
unshift (@INC, "/home/de/lib/perl/lib");
unshift (@INC, "/home/de/lib/perl/arch");
}
```

ただし，同じように見えてこの2つの動作の違っていてはuse libを使った場合はPerlがプログラムを実行するときに，文法チェックコンパイルを行うので，そのときに指定したパスがあるかどうかチェックしてくれますが，下のunshiftを使った例ではunshiftが実行されるまでパスの存在をチェックしないので，ブラウザにエラーが返るまで時間がかかってしまいます。なお，use libを使ったパスの追加方法はPerl5.002以前のバージョンでは使えません。ですので，比較的新しいバージョンのPerl5がサーバ内にある場合はuse libを使い，WindowsNT用のPerlなどでPerl5.002以前のバージョンしか使えない場合は下のUNSHIFTを使ったほうでモジュールのパスを設定することになるでしょう。

背景イメージ(gd.gif)
newFromGd(FILEHANDLE)
でGIFイメージをメモリ上に読み込み
カラーインデックスを作成
colorAllocate(255,0,0)
@queue配列
40 60 24 80 30 ……
(元データ)
読み込んだイメージ上に，
作成したカラーインデックスで
グラフの線を引く

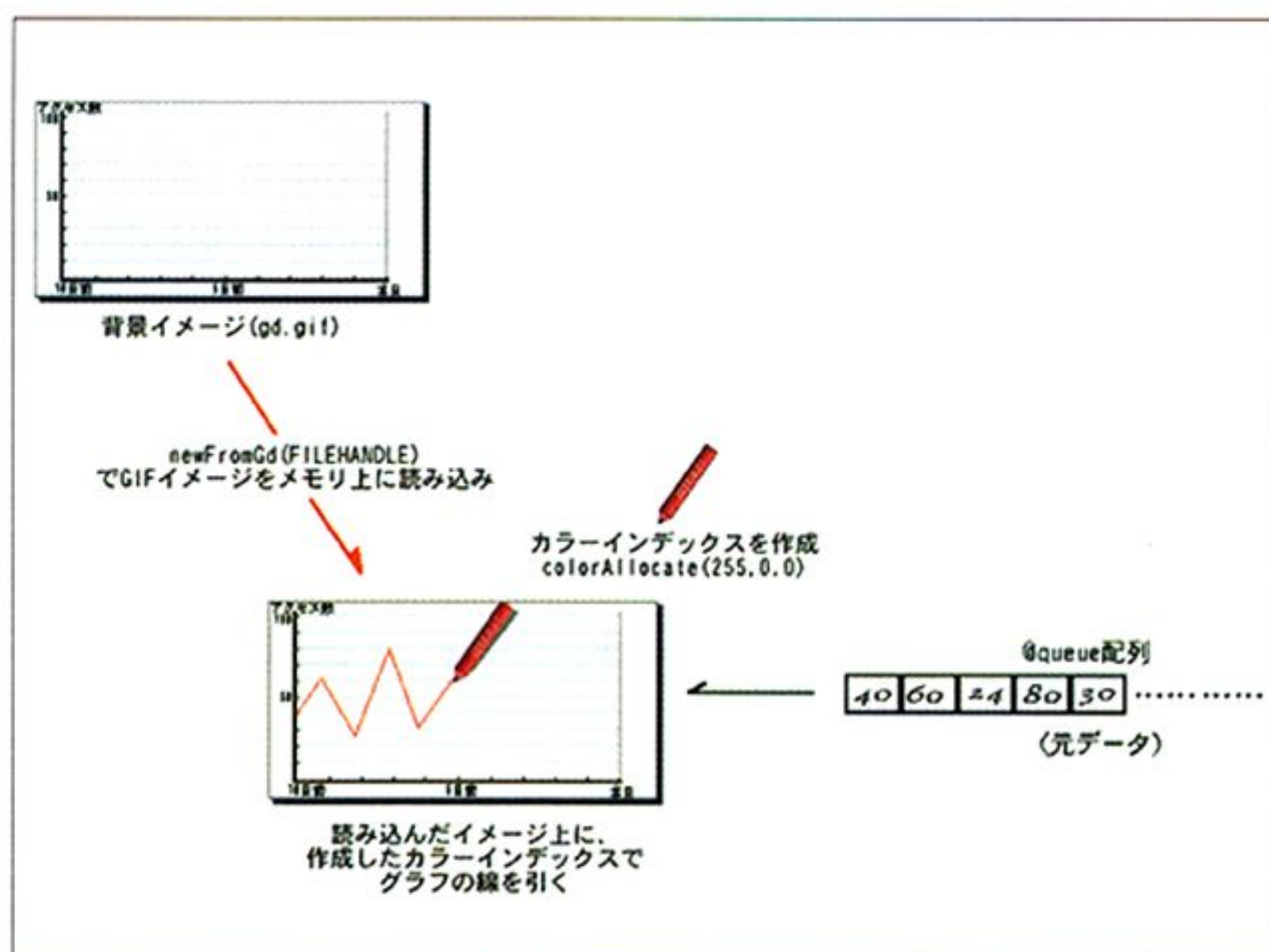



図3

GDライブラリの使い方

外部モジュールが使えるようになったら，あとは使ってみるだけですね。CPANで公開されているPerlモジュールはモジュールの内部に使用方法が埋め込まれていますので，それを見ると使い方がわかります。

Perl5がインストールされている環境であれば，

```
pod2html
pod2man
pod2latex
pod2text
```

といった実行ファイルがコマンド類のディレクトリにインストールされているはずです(たとえば筆者の使っているFreeBSDであれば，これらは/usr/local/bin/にインストールされています。

```
which pod2html
```

とすれば，もしpod2htmlがインストールされているなら，

```
/usr/local/bin/pod2html
```

とpod2htmlのフルパスが表示されます)。これらを使うとそのモジュールのドキュメントを抽出することができます。pod2の後ろに続いて書かれている形式が出力される形式です。htmlならhtmlドキュメント，textならテキスト形式で出力されます。もしその場でテキストで見たいのならば，moreやlessといったページャーがそのサーバに存在すれば，pod2textでそのまま見るのが楽かもしれません。

たとえば，/usr/lib/perl5のEnv.pmのドキュメントを見たいのならば，

```
cd /usr/local/lib/perl5
```

で/usr/local/lib/perl5に移り，

```
pod2text CPAN.pm | less
```

するとEnv.pmの解説が表示されます。先ほど，インストールしたGDライブラリのドキュメントをhtml形式で出力してみましょう。

```
cd ~/lib/perl/lib
pod2text GD.pm > GD_man.html
```

これでGD_man.htmlがファイルとして出力されます。これらのマニュアルは，すべて英語で書かれていますが形式がどのマニュアルも似通っているので，必要な情報を取り出すのはそれほど苦労しません。

だいたい，

・モジュールの名前，バージョンとこのモジュールの目的(1行)
・梗概(このモジュールを使った簡単なサンプルリスト)
/
・解説(各モジュールの簡単な概要)
・1つひとつのメソッドの説明

という順番で書かれています。だいたいにおいて，これらのドキュメントは，ちょっと使ってみる程度なら全部は読む必要はありません。勘のいい人なら，モジュールを全部読むまでもなく先頭1～2行に書かれたモジュールの紹介と梗概のリストを読むだけで使い方がわかってしまうこともあります。

ここで，サンプルプログラムがどのようにGDライブラリを使っているのかをここで簡単に解説しましょう。今回，筆者はこのGDのhtml出力ドキュメントを簡単に日本語に訳してみましたので，これも同時に見ていただけると，理解の手助けになると思います。

・簡単な仕様

このサンプルプログラムではまず，グラフの背景となるgd.gifというイメージをメモリ上に読み込みます。メモリ上に展開されたgd.gifイメージに数値データ(リスト中では@queue配列)を基にして折れ線を赤で描きます。(図3)

できあがったGIFイメージは標準出力に出力されて，ネットワークを通じてブラウザ上に表示されます。

この仕様を実現するのに必要なグラフィック関係の手順を挙げると，

1)GIFイメージをメモリに読み込む
2)赤線を引く

3)イメージを標準出力に出力する
の3つができればいい，ということになりますね。

サンプルプログラムを実装する

さて，GD.pmのマニュアルをご覧ください。
Perlモジュールはオブジェクト指向なので，まずは必要なオブジェクトをピックアップしてみましょう。
GD.pmでは3つのクラスが定義されています。

GD::Image
イメージの作成を行うクラスです
GD::Font
フォント，テキストの作成を行うクラスです
GD::Polygon
ポリゴンを作成するクラスです

マニュアルではこのように書かれています。フォント，テキストやポリゴンはここでは使いませんので，GD::Imageクラスのみ使うことで実現できそうです。ここで，下にある「簡単なサンプル」とその下の注意を読んでおくと，このクラスの使い方が理解できます。以下はマニュアルからの抜粋です。

1.新しい空のイメージを作るためには，new()メッセージをGD::Imageに対して送り，イメージの横幅，高さの情報をオブジェクトに渡してください。イメージオブジェクトが返されます。ほかのクラスメソッドでは，すでに存在するGIFやGD，あるいはXBMファイルから初期化することができます。

2.次の作業は通常，イメージのカラーテーブルに色を追加することになります。
色はcolorAllocate()メソッドコールを使って追加されます　それぞれのコールの3つのパラメータは表示したい色の赤，緑，青(RGB)成分です。メソッドはイメージ内でのカラーテーブル内における色のインデックスを値として返します　これらのインデックスを後で使うために保存しておくとよいでしょう。

3.これで，drawする(絵を描く)ことができます。数多くのグラフィックプリミティブが下に解説されています。このサンプルでは，テキストの描画，楕円の作成，ポリゴンの作成と描画を行っています。

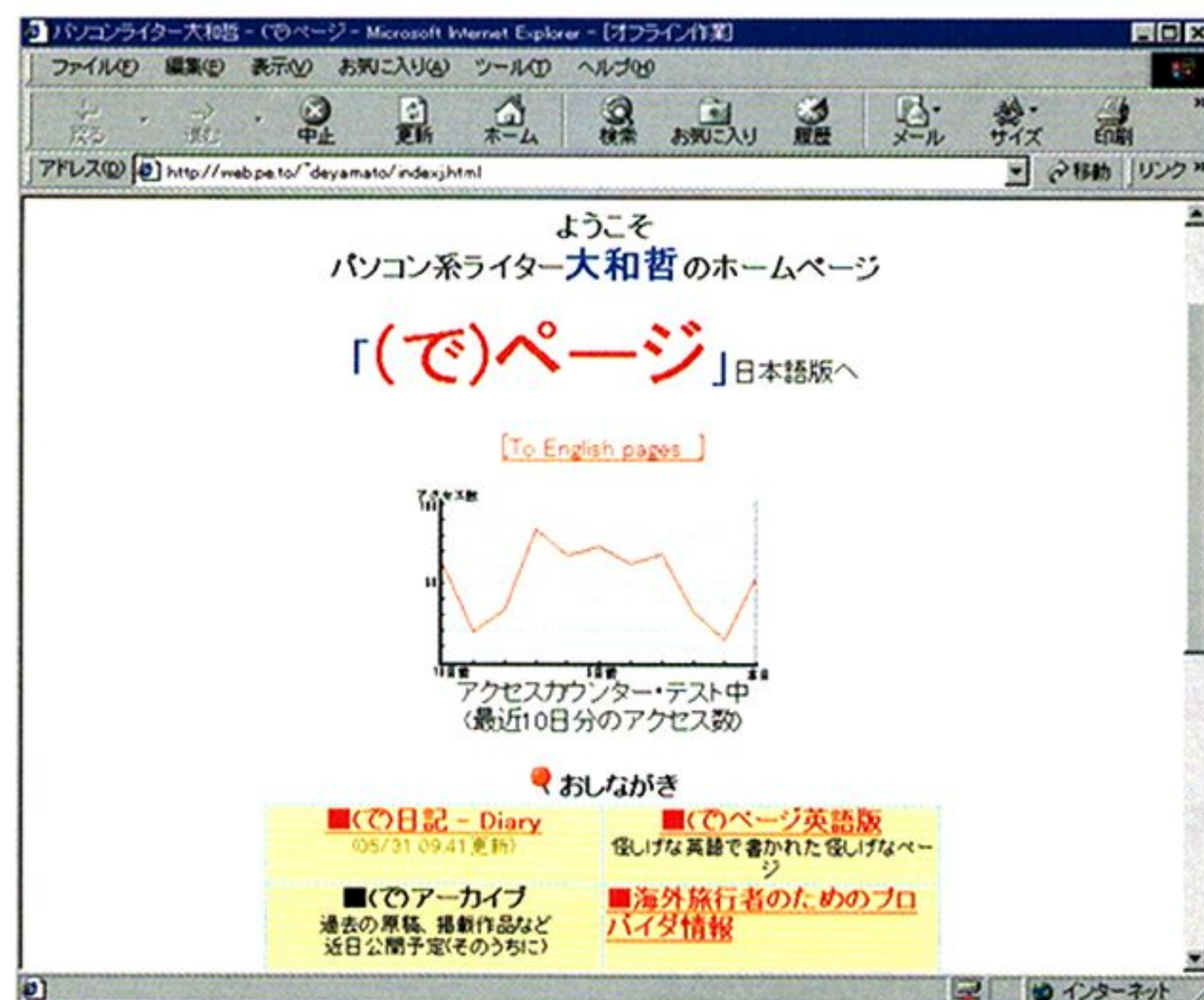



図4　アクセス者数がグラフ表示できる

さて，このクラスの初期化の方法です。普通に考えれば，

new GD::Image(100, 100)

でこのクラスの変数を新たに定義するわけですが，先ほどの注意の中に，
「すでに存在するGIFやGD，あるいはXBMファイルから初期化することができます」
とありましたね。「new」と「gif」を含むメソッドを探してみましょう。それはここにあります。

newFromGif
GD::Image::newFromGif(FILEHANDLE) class method
これはファイルハンドルで与えられたGIFファイルからイメージを作成します。ファイルハンドルはあらかじめ有効なGIFファイル，もしくはパイプなどに対してopen()されていなければなりません。もし成功した場合は，このメソッドは指定されたイメージと同じ内容で初期化されたイメージを返します。もし，失敗した場合(通常，有効なGIFファイルへのハンドルを渡されなかったり，通常に終了できなかった場合)，メソッドはundefを返します。メソッドは自動的にファイルハンドルをcloseするわけではないことに注意してください。しかしながら，このメソッドはbinmode(FILEHANDLE)は自動的に呼び出します。サイズについて，あるいは色をどのように使われているかといった情報を得るために，下に示すようなイメージ関連のクエリーメソッドを使うことができます。

```
Example usage:
open (GIF,"barnswallow.gif") || die;
$myImage = newFromGif GD::Image (GIF) || die;
close GIF;
```

通常はこのようにそのクラスの変数を新たに作り出す場合にはnewでデフォルトのコンストラクタを呼び出すわけですが，実は，そうではなくて，特殊な方法で初期化したい場合など，デフォルトとは違うコンストラクタが定義されていることがPerlモジュールにはよくありますので「梗概」やクラスの解説とnew()についている書かれているあたりはよく注意して読んでみましょう。
ともあれ，これを使うと仕様の1)は実現できますね。つづいて，2)の赤線を引く方法を考えましょう。線を引く方法ですから「line」という単語を含むメソッドを探しましょう。ブラウザの「編集」-「このページの検索」メニューで「line」という単語を含む部分を探してください。すると，

line
GD::Image::line(x1, y1, x2, y2, color) object method
このメソッドは(x1, y1)から(x2, y2)へ既に設定されたカラーで直線を引きます。

というGD::Imageのメソッドがありました。これで線を引けそうです。ここでは「赤色」で線を引きたいのでしたね。カラーテーブルに赤い色をアサインする必要があります。
これについての解説も先ほどの「注意」にありました。

「色はcolorAllocate()メソッドコールを使って追加されます　それぞれのコールの3つのパラメータは表示したい色の赤，緑，青(RGB)成分です。メソッドはイメージ内でのカラーテーブル内における色のインデックスを値として返します　これらのインデックスを後で使うために保存しておくとよいでしょう」

あらかじめ，ColorAllocateで「赤」色を定義しておき，lineで線を引けば2)も目的は達成されますね。
そして，最後，標準出力への出力です。ここはこんな感じに書かれています。

5.描画を完了したら，あなたはイメージを，gif()メッセージを送ることによってGIFフォーマットにコンバートできます。この動作で，オブジェクトはイメージのバイナリデータを含んだスカラー値(かなり大きいかもしれません)を返します。これを通常はスクリーンに出したり，ファイルに保

リスト1

```
#!/usr/local/bin/perl

# set your user libraly directory
use lib '/home/deyamato/lib/perl/lib','/home/deyamato/lib/perl/arch';
use GD;

my @queue;#整数11個の配列
&Getlog();

print "Content-type: image/gif\n\n";

# create a new image
#$im = new GD::Image(100,100);

open (GIF, "bg.gif") || die "Not open";
$im = newFromGif GD::Image(GIF) || die;
close GIF;

# allocate some colors
$white = $im->colorAllocate(255,255,255);
$red = $im->colorAllocate(255,0,0);

# make the background transparent and interlaced
$im->transparent($white);
$im->interlaced('true');

#線を描くi
my $px,$py;

($dummy,$py)=split(/,/,$queue[10]);
$py=109-$py;
for($dx=1;$dx<11;$dx++){
        $_=$queue[10-$dx];
        ($dummy,$dy)=split/,/;
        if($dy eq ""){ $dy=0;}
        $px=($dx-1)*20+17;
        $cx=$dx*20+17;
        $cy=109-$dy;
        if($cy<0){$cy=0};
        $im->line($px,$py,$cx,$cy,$red);
        $py=$cy;
}

binmode STDOUT;

# Convert the image to GIF and print it on standard output
print $im->gif;

exit(0);

sub Getlog{
        my $date,count;
        my $count=0;
        open(LOG,"logfile.log")||die "ログファイルが読み込めません\n";
        while(<LOG>){
                chop;
                &put_queue($_);
                ($date,$count)=split/,/;
                #print "[$date] $count \n";
        }
        close LOG;

        #現在の日付を取る
        ($sec, $min, $hour, $day, $mon, $year)=localtime;
        $mon ++;
        $day = sprintf("%.2d", $day);
        $mon = sprintf("%.2d", $mon);
        my $current_date;
        $current_date="$year$mon$day";

        #--ログの最後の日付をチェック
        if(0==($current_date cmp $date)){
                $count++;
                $queue[0]="$current_date,$count";
        }else{
                put_queue("$current_date,1");
        }

        #-- ログファイルにかき出す
        open(LOG,">logfile.log");
        for($i=10;$i>=0;--$i){
                if($queue[$i] != ""){
                        print LOG "$queue[$i]\n";
                }
        }
        close(LOG);

}

sub put_queue{
        #print " ** put_queue!! **\n";
        for($i=10;$i>0;$i=$i-1){
                $j=$i-1;
                #print "queue$j($queue[$j]) => queue$i($queue[$i])\n";
                $queue[$i]=$queue[$i-1];
        }
        $queue[0]=$_[0];

}
```

管したりするわけです。

ということでした。つまるところCGIでは,

print "Content-Type: text/html¥n¥n";

を出力してから,

GD::Img::gif ()

を呼び出すことで，CGIからGIFイメージを出力することができるというわけなのでした。

終わりに

ということで，サンプルプログラムのような折れ線グラフ式アクセスカウンター，gd1.cgiができました。いま，筆者のホームページでは，このカウンターをつけて毎日どのくらいの人が訪れるのかをグラフで表示しています。実際に動いているところをご覧になりたい方はこちらへどうぞ。

http://www.web.pe.to/deyamato/

なお，このcgiはGIFイメージそのものを出しますので，ホームページ上で使うには，このグラフを出したいページのHTMLファイルに,

<IMG SRC="gd1.cgi">

と書くだけで，イメージをページに表示できます。また，GIFグラフィックイメージを出すのですから，当然，ブラウザのアドレス(URL)欄に,

http://www.foo.com/gd1.cgi

とURLを打ち込んでもそのままブラウザにグラフィックが表示されます。クライアント側からの扱いは，普通のgifファイルと全然変わりがないんですね。

さて，Perlのモジュールというのは，すべてをつぶさに読むまでもなく，簡単に使おうと思うのなら，「クラスの解説」部分と「サンプルリスト」を読むだけで大方のめどはついてしまうのですね。もちろん，実際に使いたい機能は，このページ内検索してメソッドの解説を読んでおいたほうがいいですが，いずれにしても英語ドキュメントを目の前にしてもうんざりすることはありません。必要なのはほんの一部だけなのです。

CPANにはGDのほかにも，さまざまなライブラリがあります。グラフィックだけではなく，ファイル関係，検索エンジンに使えるユーザーエージェント(Webロボット)などもあります。どれもドキュメントはほとんどが英語ですが，要領さえつかめばこれらのモジュールを使うのはそんなに難しいことではありません。もちろん，すべてを使うことができるわけではありませんが，一般ユーザーでも,

use lib

でライブラリのパスを定義すれば使えるものがたくさんあります。ぜひ，皆さんも，プロバイダやホームページを置いているサーバが，Perlライブラリを使えるような環境であるならば，一度はCPANにあるライブラリを使ったCGIを使ってみてください。きっと，皆さんにもいままでにはなかった新しいCGIのアイデアが湧いてくるのではないかと思います。

参考文献

GDライブラリ　Ver1.19マニュアル
Lincoln D. Stein (1995)

プログラミングperl 2ndエディション

壁紙＆サウンドチェンジャを作る

菊地 功　Kikuchi Isawo

昨今ではパーソナルプログラム開発の主力であるVisualC++入門が基本解説ばかりではもの足りないということで，ここでは少しVC++入門の応用編だ。実際に少し具体的なアプリを作っていってみよう。今回は壁紙チェンジャを制作する。

おや？と思った人，大正解。今号はこの記事は2つあるのでした。涙なくしては語れない，海よりも深い深い事情が……あるわけじゃなくて（編注：あったりして……。春号でVC++の原稿が2つあったということをコロっと忘れてたので……）。セクション1では，どちらかというと王道的なアプリケーションを手掛けた。ビューがあって，そこにユーザーがドキュメントをがりがりと作成していくタイプだ。

セクション2では，Tips的な小技を交えた，小物アクセサリに挑戦してみよう。ずばり壁紙オートチェンジャ。ありがちだけどね。まあ豊富な機能をたくさん盛り込むのはおのおのに任せるとして，必要最小限の機能だけをきっちり仕上げたい。一定時間ごとに特定フォルダ内からランダムに画像を抽出し，それを壁紙にする。これだ。ただそれだけでは芸がないので，付加価値として，サウンドチェンジャ機能もつけてみよう。Windowsのサウンド，みんな自分で設定したりしてるかな。筆者はテレビからの録音や，ゲームからの吸い出しなどでシリーズ化しているのだが，その数がマシンの所有台数を超えてしまった。そんな場合には最適だね。オンラインソフトでそういった機能を持つものを探してみたのだが，筆者が見る限り「一定時間ごとに自動で」というものはなかった。需要がないのかもしれないけど。そういったわけで，壁紙と同時にサウンドを変える機能をつけてみる。

基本の流れを作ってみる

では，プロジェクトを作ろう。だだっ広いビューは必要ないが，ダイアログだけってのも貧相なので，今回はフォームビューというものを使ってみよう。フレームは普通のウィンドウと同じなのだが，中身はダイアログテンプレートという，ちょっと変わったウィンドウだ。

プロジェクトの新規作成で，いつものようにMFC AppWizard（exe）を選択し，プロジェクト名はwchangeとでもしておこうか。アプリケーションの種類はSDIにし，ステップ4でツールバーもステータスバーも印刷も必要ないのでチェックを外す（図1）。また，詳細設定ボタンを押して，今度はウィンドウスタイルタブを開いてもらいたい。ここで，「メインフレームのスタイル」枠の中の「サイズ変更」と「最大化ボタン」のチェックは外しておく（図2）。そして最後，ステップ6ではCWchangeViewクラスを選択して，基本クラスから“CFormView”を指定する（図3）。これでOKだ。

プロジェクトができたら，まずはリソースの中を見てみよう。Dialogリソースの中に，IDD_WCHANGE_FORMというダイアログテンプレートができているはずだ。これが，ビューにはまるテンプレートになる。いままでのダイアログテンプレートと違ってタイトルバーなどがついていないが，気にしなくてよい。まずは，このテンプレートに図のようにコントロールを載せよう（図4）。「壁紙ファイルの場所」を示すエディットボックスのIDはIDC_FOLDER，参照ボタンはIDC_BROWSE，時間間隔のエディットボックスはIDC_INTERVAL，スピンボタンはIDC_INTERVALSPINとしておこう。スピンボタンはセクション1と同じように自動関連付けにしていく。あと，OKボタンのIDはIDC_OKとし，スタイルで「標準のボタン」にチェックをしておくこと。この「標準のボタン」というのは，（リターンキーを受けないコントロールがアクティブなときに）リターンキーを押したときにクリックされたことになるボタンで，ボタンの枠が太く表示される。デザインができたら，コントロールとメンバ変数をマップしよう。ClassWizardのメンバ変数タブで，クラス名はCWchangeViewを選択する。変数の登録は，IDC_FOLDERとIDC_INTERVALの2つのエディットボックスと，スピンボタンだけでいい。ここで，IDC_FOLDERはフォルダ名が入るのでCStringに，IDC_INTERVALは時間（分単位）が入るだけなので，intにしておいていいだろう。範囲は1分から，最長120分くらいにしておこうか（図5）。そうしたら，このままメッセージマップのほうへ行って，IDC_BROWSEボタンがクリックされたときと，IDC_OKボタンが

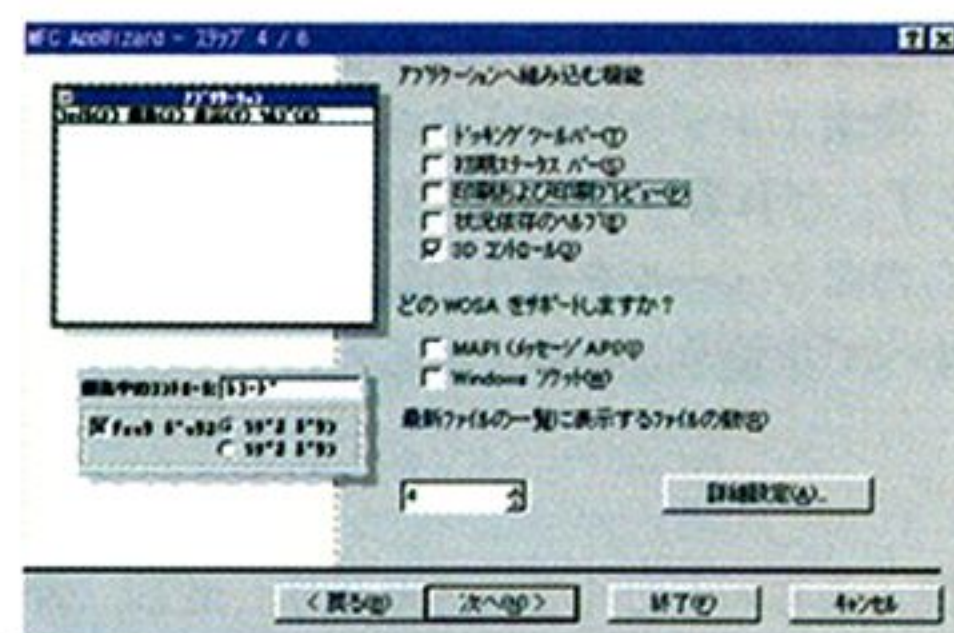



図1　ステップ4

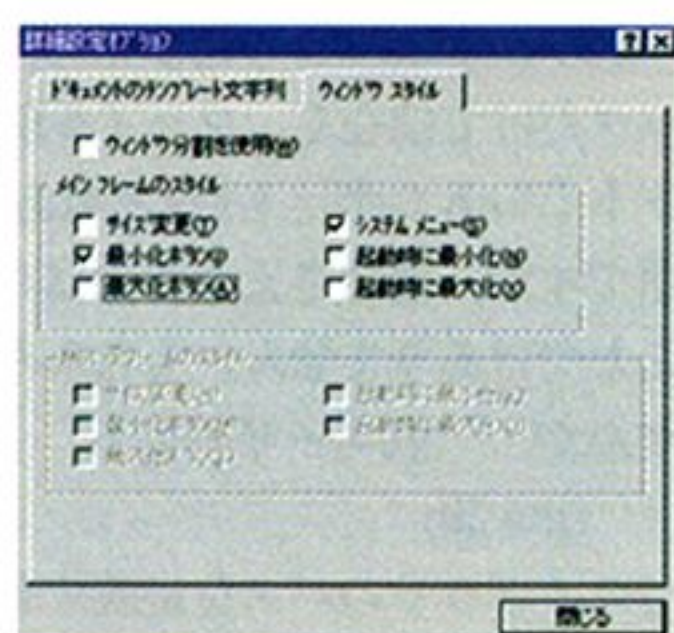
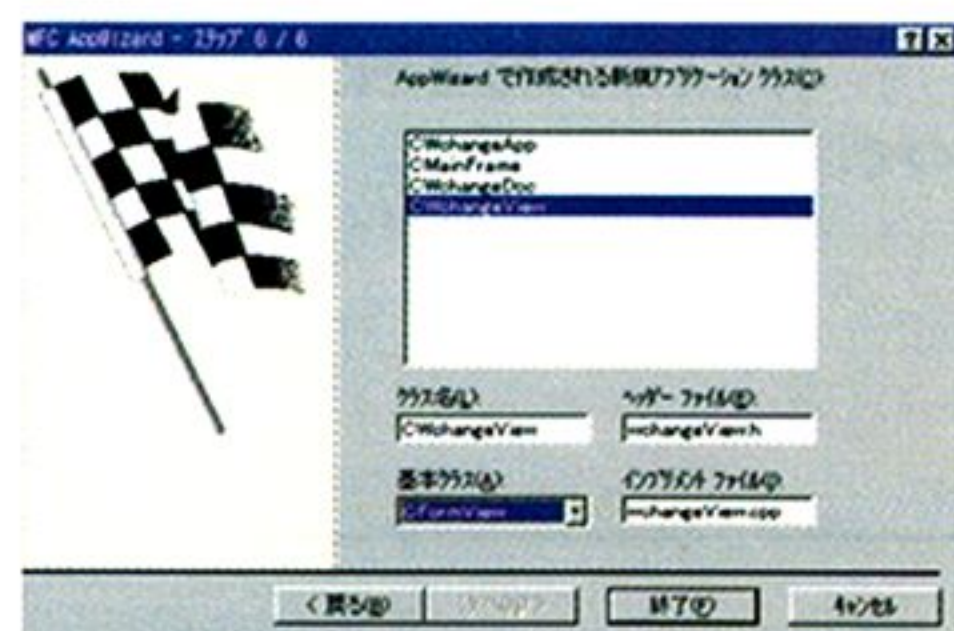
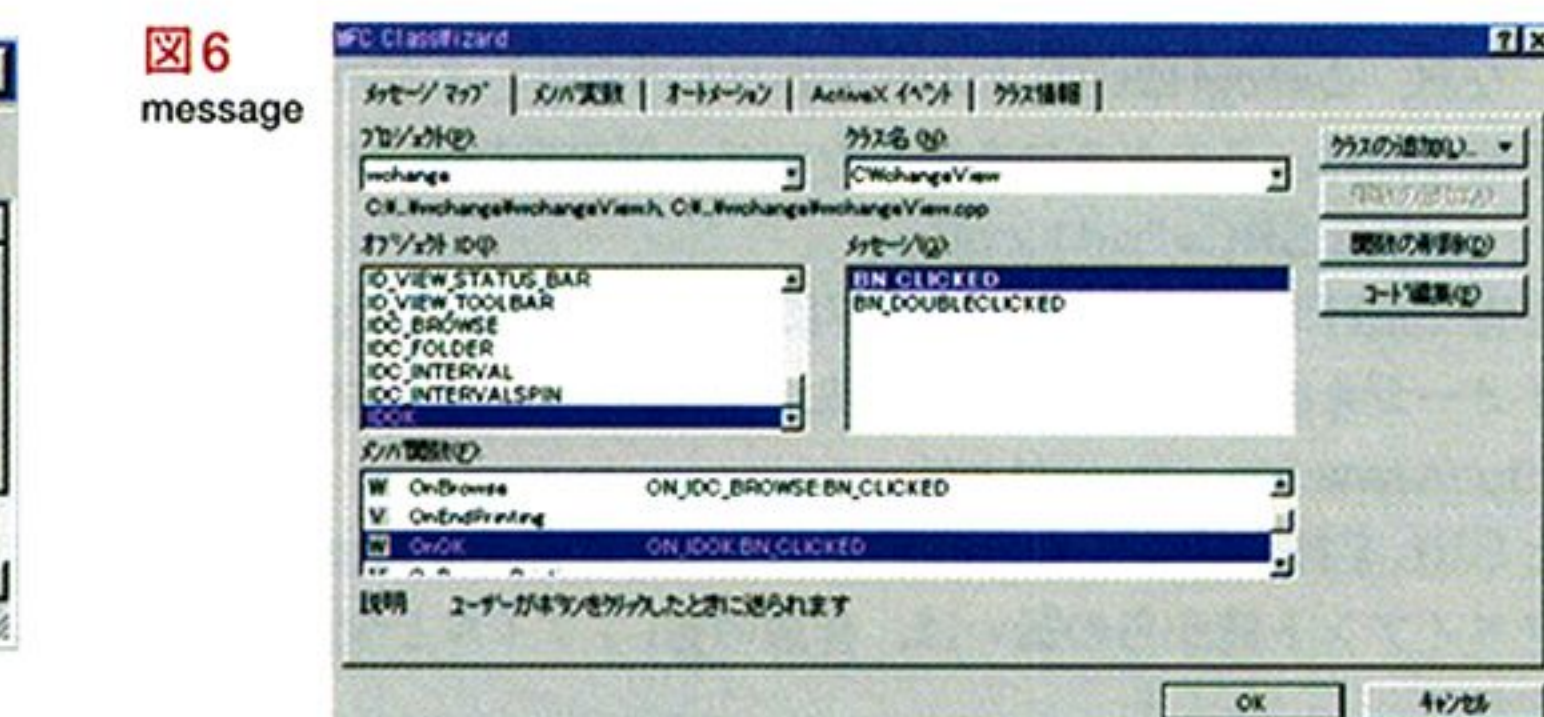
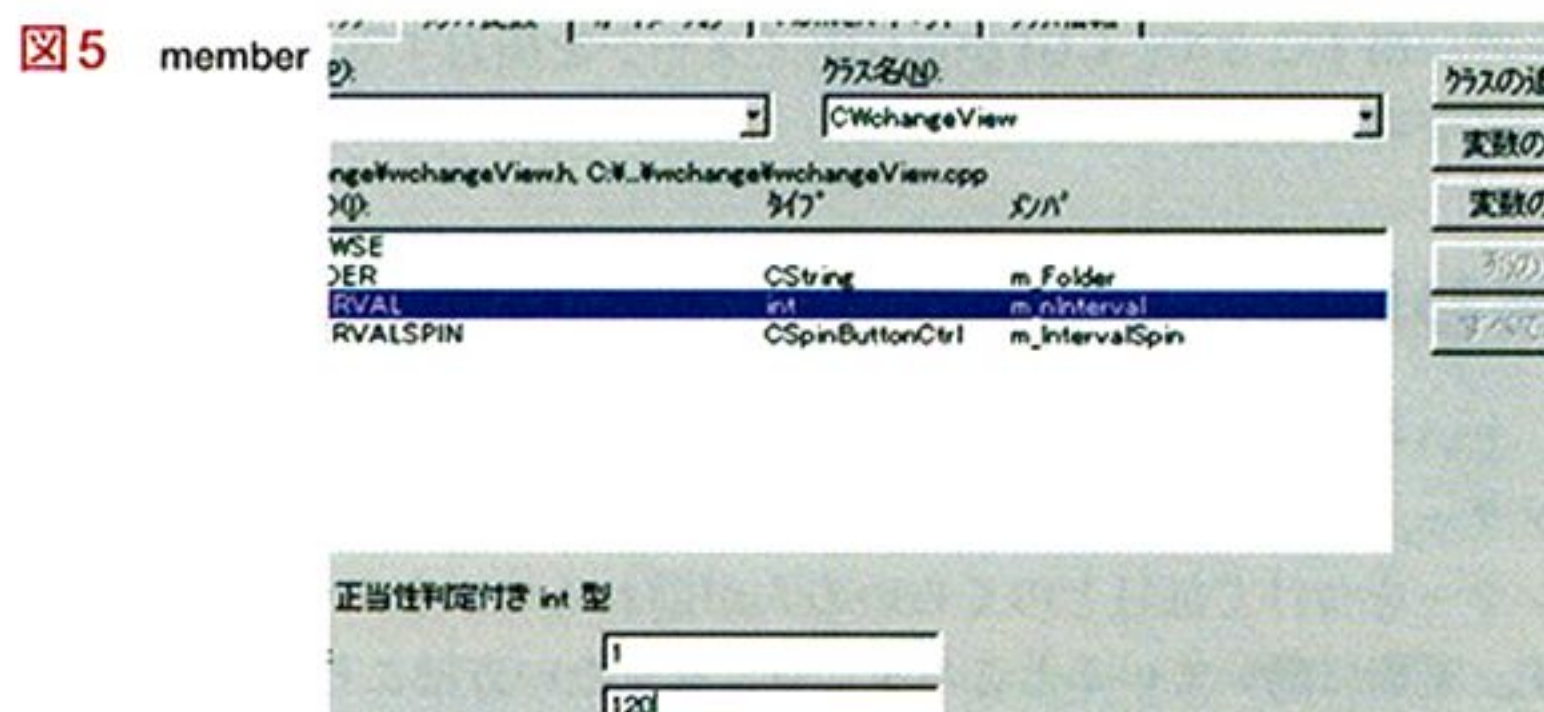



図2　詳細設定

図3　ステップ6

図4　form

図5　member

図6　message

クリックされたときのメッセージ関数を作ってしまおう(図6)。

まずはOnBrowse()メソッドだ。ここでやることは，ファイルダイアログを開いてフォルダをユーザーに選択させ，指定されたフォルダ名をIDC_FOLDERのエディットボックスに入れてやる。ファイルダイアログは，CFileDialogというクラスがすでに用意されているので，それを使おう。フォルダを指定させるのに，タイトルバーに「ファイルを開く」とかついてしまうが，気にしないことにする。次のような感じだ。

```
CFileDialog dlg( TRUE, NULL, "SelectFolder",
OFN_HIDEREADONLY|OFN_PATHMUSTEXIST,
 "BitmapFiles (*.bmp)|*.bmp||", this );
if( IDOK==dlg.DoModal() ){
  CString str = dlg.GetPathName();
  int count = str.ReverseFind( '¥¥' )+1;
  if( count>0 ) m_Folder = str.Left( count );
  else m_Folder = str;
  UpdateData( FALSE );
}
```

CFileDialogはコンストラクタで初期化している。第1引数はTRUEで「ファイルを開く」ダイアログ，FALSEで「ファイルの保存」ダイアログとなる。第2引数は，省略した場合に付加される拡張子だが，ここでは関係ないのでNULLでよい。第3引数は，ダイアログを開いたときに「ファイル名」のエディットボックスに初期表示してやる文字列だが，今回はフォルダを拾いたいだけなので，これはダミーだ。その次はフラグ。OFN_HIDEREADONLYはリードオンリー属性を持つファイルは表示しない，ということではなく，ダイアログに「読み取り専用ファイルとして開く」チェックボックスをつけないという意味だ。あとはOFN_PATHMUSTEXISTくらいをつけておけばいいだろう。こちらは見たまんま，パスが存在しなければならない，というフラグだ。

次の文字列は，ファイルリストに表示されるファイルのフィルタだ。「ファイルの種類」リストに加える文字列と，それを選択したときにフィルタリングされるワイルドカード，2つを1ペアとし，"¦"で区切った文字列で表す。今回は1ペアだけだが，複数ある場合はどんどん"¦"でつなげていけばよい。終端はそれとわかるように"¦"を2つ重ねる。

このダイアログはモーダルでぜんぜん構わないので，DoModal()で表示してやろう。戻り値IDOKは，ユーザーがちゃんとファイルを選んで(いまの場合はフォルダだが)OKした場合の戻り値だ。もしキャンセルしたのなら，IDCANCELが返る。ここで，もしOKされたのなら，GetPathName()で選択されたファイルのフルパスが返ってくる。しかし，そのフルパスの最後にはダミーのファイル名"SelectFolder"(あるいはユーザーがなにか指定したならそのファイル名)がくっついているので，文字列の後ろから"¥"を検索し，それ以前の部分をm_Folderに入れてやる。最後に忘れずにUpdateData(FALSE)してやれば，無事にエディットボックスにパス名が取得できるというわけだ。

OKボタンのほうはとりあえず置いといて，先に設定項目のレジストリへの読み書きをやってしまおう。春号の復習だ。今度はダイアログではないのでOnInitDialog()メソッドはないが，代わりとしてOnInitialUpdate()でもオーバーライドして，そこで読み込んでやればいい(ついでにスピンボタンの初期化も)。書き込みも，春号ではDestroyWindow()でやったが，どうやらビュークラス(あるいはCFormView)は，ウィンドウの破棄をDestroyWindow()でやっていないらしい。このあたりはMFCがやっていることなのでよくわからないが，とにかくDestroyWindow()には処理が回ってこないので，WM_DESTROYのメッセージ関数でやることにしよう(この辺りは割と試行錯誤だ)。

```
void CWchangeView::OnInitialUpdate()
{
  CFormView::OnInitialUpdate();

  CWinApp *pApp = AfxGetApp();
     m_Folder = pApp->GetProfileString( "Settings",
                                        "Folder", NULL );
  m_nInterval = pApp->GetProfileInt( "Settings", "Interval", 5 );
  m_IntervalSpin.SetRange( 1, 120 );
  UpdateData( FALSE );
}

void CWchangeView::OnDestroy()
{
  CFormView::OnDestroy();

  UpdateData( TRUE );
  CWinApp *pApp = AfxGetApp();
  pApp->WriteProfileString( "Settings", "Folder", m_Folder );
  pApp->WriteProfileInt( "Settings", "Interval", m_nInterval );
}
```

m_Folderのほうは文字列なのでString系のメソッドを使うが，使い方はInt系と同じだ。時間間隔は初期値で5分くらいにしておこうか。あと，アプリケーションを終了しないままWindowsを終了させた場合だが，その場合でもビューにはちゃんとWM_DESTROYメッセージがくるようだ。春号のダイアログには確かにWM_DESTROYもこなかったのに(この辺も試行錯誤)。というわけで，WM_ENDSESSIONはハンドルする必要はない。

さて，OKボタンが押されたときから，タイマをカウントすることにする。これには，CWnd::SetTimer()という，そのものずばりのメソッドがあるので，それを使うことにしよう。このメソッドは，指定した時間ごとにウィンドウにWM_TIMERメッセージを送るか，あるいは指定した関数を呼び出すことができる。

```
UpdateData( TRUE );
if( m_nIDEvent ) KillTimer( m_nIDEvent );
m_nIDEvent = SetTimer( 1, m_nInterval * 60 * 1000, NULL );
AfxGetMainWnd()->ShowWindow( SW_MINIMIZE );
```

あらかじめm_nIDEventというメンバをCWchangeViewに作っておく。これは，タイマのIDで，SetTime()の第1引数と同じ値が返ってくるはずだが，念のために変数としている。また，SetTimer()に渡すIDのほうは，0以外であればなんでもいいようだ。もちろん複数のタイマを並列して動かす場合には，別の値を使う。第2引数はミリ秒単位の時間，第3引数はタイムアウト時に呼ばれる関数を指定するが，今回はメッセージで処理するので指定しなくてよい。念のため，SetTimer()でタイマを動かす前に，もしすでにタイマが動いていたらKillTimer()で止めておこう。ついでに，タイマを設定したらウィンドウを最小化しておけば，邪魔にならないだろう。

正常にSetTimer()が働いたなら，指定時間ごとにWM_TIMERがビューに送られる。これをハンドルして，そこで壁紙を変更しよう。壁紙の変更には，SystemParametersInfo()というWin32APIを使うが，その前に変更すべき壁紙を指定されたフォルダ内から選択しなければならない。

特定のフォルダ内のファイルを取得する方法はいくつかあるが，今回はWin32APIのFindFirstFile()とFindNextFile()を使ってみる。前者はフォルダ内の最初のファイルを取得する関数で，後者は前者の戻り値で取得したハンドルを使って，次々とファイル情報を取得する関数である。基本的な使い方は，最初にFindFirstFile()を1回だけ呼び出し，それ以降FindNextFile()をファイルがなくなるまで(戻り値がFALSEになるまで)呼び出すというかたちだ。また，最後には忘れずにFindClose()でハンドルを閉じておく必要がある。これらの関数を使って，まずはBMPファイルの数を数えてみよう。

```
CString path = m_Folder+"*.bmp";
WIN32_FIND_DATA *pFindData = new WIN32_FIND_DATA;
```

```
HANDLE hFile = FindFirstFile( (LPCTSTR)path, pFindData );
int count = 0;
if( hFile!=INVALID_HANDLE_VALUE ){
  count++;
  while( FindNextFile( hFile, pFindData ) ) count++;
  FindClose( hFile );
}
if( count<2 ) return;
```

フォルダの場所を指定したエディットボックス内の文字列に"*.bmp"を付加し，FindFirstFile()で検索をかける。もし最初のファイルが見つかったら，FindNextFile()でファイルがなくなるまで検索し，その回数をカウントする。これでBMPファイルの数を取得できるはずだ。もっとスマートにファイル数を取得できる関数があってもよさそうなものだが，とりあえずこれで間違いはない。もしBMPファイルが1個以下ならば，チェンジャの意味を成さないので終了する。

さて，今度は見つけたもののうち，何番目のファイルを使うかを決定する。ランダムに選択させるため，rand()という疑似乱数を発生させる関数を使用しよう。rand()%countとすれば，0からcount－1までの数値が(ほぼ) 等確率で出現するはずだ。ただし，このrand()関数を呼び出す前に，前もってsrand()という乱数を初期化する関数を呼んでおく必要がある。これは最初に1回呼んでおけばいいので，コンストラクタで記述しておけばいいだろう。引数として乱数のシードを取るが，これは時間を放り込むのが一般的だ(同じシードを指定すると，毎回同じ乱数が現れる)。GetTickCount()あたりが引数もなく，戻り値として値を返すので簡単だろう。ただ，このGetTickCount()は時間は時間でも，Windowsが起動してからのミリ秒単位の時間を返す。スタートメニューにでも登録した場合，毎回ほぼ同じ値を返してしまうが，まあミリ秒単位で毎回まったく同じ値になるとも思えないので，気にしないことにする。気になる人は，ちゃんとした時間取得関数とでも組み合わせてもらいたい。

話がそれたが，何番目のファイルを使うかを決定したら，またFindFirstFile()とFindNextFile()を使って，目的のファイルまでループさせ，SystemParametersInfo()で壁紙としてそのファイル名を指定する。

```
count = rand()%count;
hFile = FindFirstFile( (LPCTSTR)path, pFindData );
while( count-- ) FindNextFile( hFile, pFindData );
FindClose( hFile );
path = m_Folder+pFindData->cFileName;
SystemParametersInfo( SPI_SETDESKWALLPAPER, 0,
(PVOID)(LPCTSTR)path,
SPIF_UPDATEINIFILE|SPIF_SENDCHANGE );
```

なかなかカウントのしかたがトリッキーだが，よく考えて理解してもらいたい。この辺を自在に使えるようにならないと，一人前にはなれないぞ。

さて，ファイル名はというと，FindFirstFile()なんかで渡していたWIN32_FIND_DATA構造体のcFileNameメンバに入れられる。ただし，これにはパス名がついていないので，またm_Folderと連結して，フルパス名にする。それをSystemParametersInfo()に渡すわけだが，これは別に壁紙を変更するため専用の関数ではなく，システム周りのさまざまな設定や情報の取得のための関数だ。第1引数にSPI_SETDESKWALLPAPERを指定し，第3引数にファイル名を指定することで，そのファイルを壁紙にしてくれる(第2引数は無視される)。最後の引数は情報を変更したあとの動作を示すフラグで，SPIF_UPDATEINIFILEはイニシャルファイルを更新することを(この関数で設定できる項目は，SYSTEM.INIというファイルに記述されている，が，レジストリにも入っている)，SPIF_SENDCHANGEは設定が変更されたことをほかのウィンドウにメッセージとして送ることを示す。両方設定しておけば，間違いはないだろう。

ところで，SystemParametersInfo()のヘルプを見ると，「壁紙の設定」項目はあるが，「並べて表示」「中央に表示」といった設定項目はない。今回はこの辺のことはやらないが，自分で拡張する場合のために，話だけはしておこう。これらの変更はレジストリを書き換えることで行える。レジストリエディタで，HKEY_CURRENT_USER¥control panel¥desktopのキーを開いてもらいたい。ここに，TileWallpaperとWallpaperというのがあるだろう(図7)。Wallpaperに設定されている文字列が，いうまでもなく現在設定されている壁紙のファイル名で，TileWallpaperのほうは，これが"1"(数値ではなく文字列)のときは「並べて表示」，"0"のときは「中央に表示」であることを示している。また，もしPlus!がインストールされていれば，WallStyleという項目に"2"を指定することで，「画面全体に表示」となる。したがって，まずここでTileWallpaperなりWallStyleを変更しておいて，SystemParametersInfo()で壁紙を変更すれば，表示位置も変更できる。なお，レジストリを書き換えただけでは，システムは多分それに気づかず，反映はされないだろう。任意の場所のレジストリの書き換えについては，サウンドチェンジャのところで触れる。

なにはともあれ，まずは壁紙チェンジャとしては機能するようになったはずだ(サンプルwchange1)。起動して，適当なフォルダと時間を指定からOKを押し，しばら～くぼーっとしてると，壁紙が変わる……のはいいのだが，ウィンドウが無駄にでかかったり，タイトルバーに「無題」とか書いてあったり，余計なメニューがたくさんあったりと，なんか随所で間が抜けている(図8)。あと，OKを押さなければタイマが働き出さないというのでは，スタートアップに入れてWindowsに常駐させる場合に不便だ。

細かい不具合の修正

では端から片づけていこう。まずはウィンドウのフレームをテンプレートに合わせる。CFormViewってのは実はCScrollViewからの派生クラスで，したがってセクション1でも使ったResizeParentToFit()メソッドが使える。テンプレートのサイズなんて画面に比べたら小さなもんだから，なにも考えずにフレームをリサイズして大丈夫だ。しかも，CFormViewは最初からテンプレートのサイズがビューのサイズに設定されている。結果，OnInitialUpdate()内に，

```
ResizeParentToFit( FALSE );
```

の1行を加えるだけで，万事解決。はい，次の方。

「タイトルバーにドキュメント名がついちゃうんですけど……」

図7 regedit

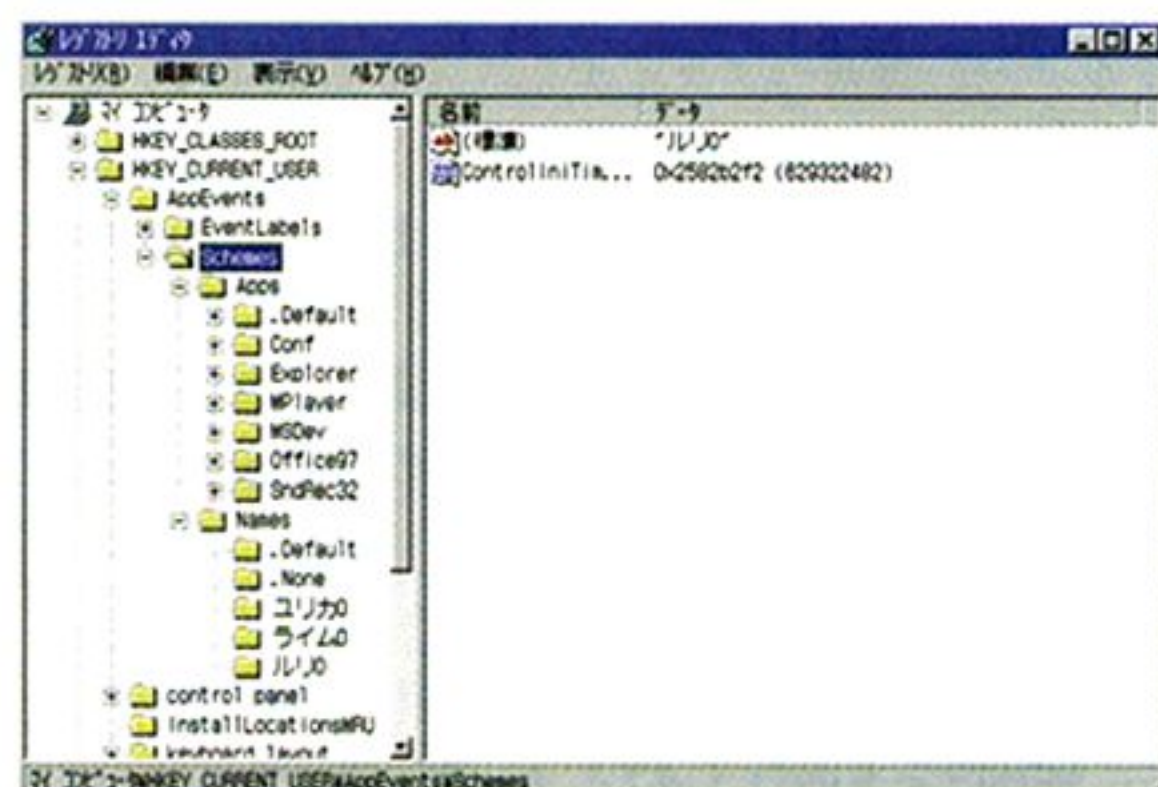
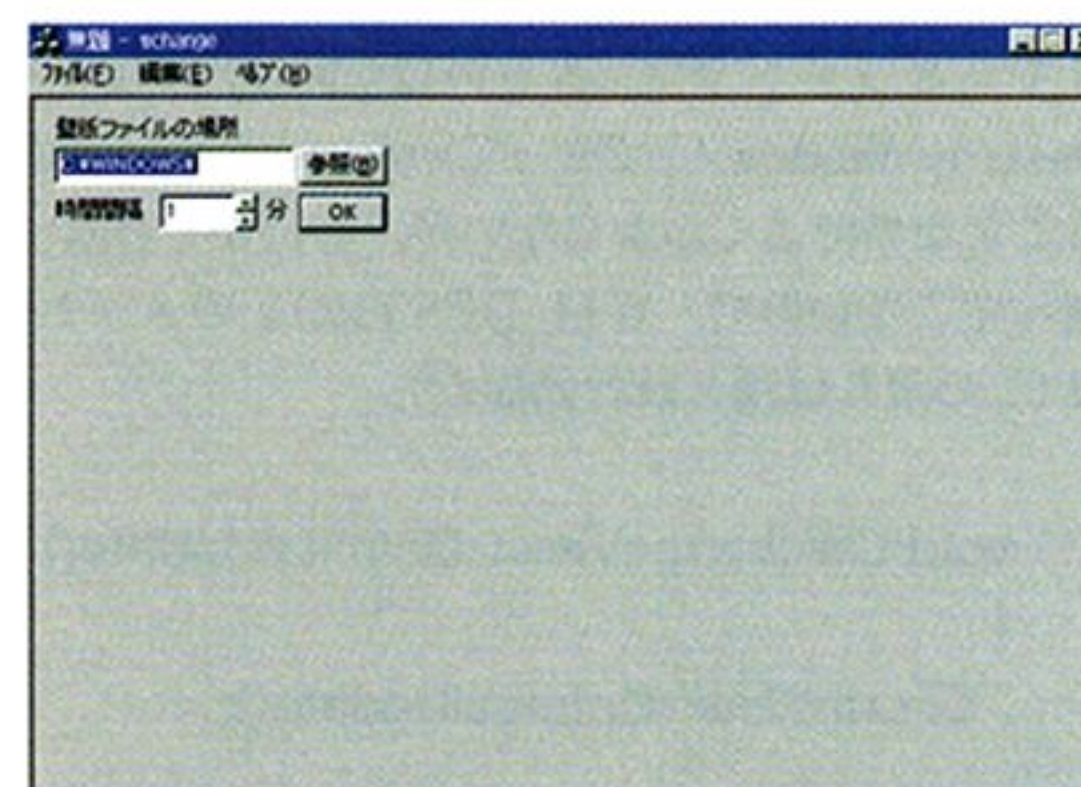



図8 wchange-1

それはね，CMainFrameのPreCreateWindow()内でスケルトンが勝手にフラグをつけてるから。

```
cs.style = WS_OVER      LAPPED | WS_CAPTION |
FWS_ADDTOTITLE
 | WS_SYSMENU | WS_MI      NIMIZEBOX;
```

のうち，FWS_ADDTOTITLEを削ってやりましょう。はい，次。
「ファイルとか編集とかのメニュー，使わへんやんけ」
これはもう元から絶っちゃいましょう。終了はシステムメニュー(ウィンドウ左上のアイコンをクリックして出るメニューね)でも，クローズボックスでもできるし。[バージョン情報]だけをシステムメニューに移しちゃいましょう。それではまず，CMainFrameクラスのLoadFrame()辺りをオーバーライドします。ここでおもむろに

```
BOOL result = CFrameWnd::LoadFrame(nIDResource,
dwDefault   Style,
 pParentWnd, pContext);
CMenu * pmenu = GetSystemMenu
                    ( FALSE                          );
pmenu->AppendMenu( MF         _SEPARATOR );
pmenu->AppendMenu( MF_STRING, ID_APP_ABOUT,
                                 "バージョン情報 (&A)..." );
pmenu->DeleteMenu( SC_SIZE, MF_BYCOMMAND );
pmenu->DeleteMenu( SC_MAXIMIZE, MF_BYCOMMAND );
CMenu * pMenu = GetMenu();
SetMenu( NULL );
pMenu->DestroyMenu();
return result;
```

とします。
いまの場合はスーパークラスのLoadFrame()を呼ばないわけにはいかないけれど，それを呼ぶまではシステムメニューも作られていないので，まずはBOOL型変数で引数を保存しておこう。次にシステムメニューのポインタを取ったら，セパレータを噛ませてバージョン情報を追加，いらないので[サイズ変更]と[最大化]もついでに取っちゃいましょ。SetMenu(NULL)は，(システムじゃない普通の)メニューをNULLに置き換えて，つまり外してしまえということね。メニューをつけるのを防ごうと思ったら，MFCのかなり深いところまで潜っていかなきゃならないから，つけたあとに外しちゃったほうが簡単でしょ。元々のメニューは削除しちゃっていいけど，ウィンドウに設定されたままじゃ削除できないから，外したあとでね。
ただし，バージョン情報をシステムメニューに入れちゃうと，今までWM_COMMANDで送られてたメニューコマンドがWM_SYSCOMMANDになっちゃいます。イメージでいうと，

```
SendMessage( WM_COMMAND, ID_APP_ABOUT, 0 );
```

だったのが，

```
SendMessage( WM_SYSCOMMAND, ID_APP_ABOUT, 0 );
```

になっちゃうってこと。だから，WM_SYSCOMMANDをハンドルして，もしIDがID_APP_ABOUTだったら，WM_COMMANDで送り直しちゃうのが簡単だね。WM_SYSCOMMANDはなぜかClassWizardではハンドルできないので，MainFrame.hの，

```
//{{AFX_MSG(CMainFrame)
```

と，

```
//}}AFX_MSG
```

の間に，

```
afx_msg void OnSysCommand( UINT nID, LPARAM lParam );
```

を，MainFrame.cppの，

```
//{{AFX_MSG_MAP(CMainFrame)
//}}AFX_MSG_MAP
```

の中に，

```
ON_WM_SYSCOMMAND()
```

を追加して，

```
void CMainFrame::OnSysCommand( UINT nID, LPARAM
                                              lParam )
{
  if( nID==ID_APP_ABOUT ) SendMessage( WM_COMMAND,
                                                   nID );
  else CFrameWnd::OnSysCommand( nID, lParam );
}
```

とでもしておけばいいだろう。はい，お大事に。
「あ，あの，起動したときに，タイマ動き出してほしいんですけど」
んなもんは，CWchangeView::OnInitialUpdate()の最後ででも，

```
m_nIDEvent = SetTimer( 1, m_nInterval * 60 * 1000, NULL );
SendMessage( WM_TIMER );
```

としとけばええやんけー！はっ！いかんいかん，ちょっとトリップしてしまった。
これでおおかた綺麗に収まったと思うけど，もう少しだけ。最小化されているときには，タスクバーじゃなくて，システムトレイにアイコンを入れたい。これにはちょっと段階が必要だ。まず，ウィンドウが最小化や元に戻されたときのメッセージをハンドルし，ウィンドウを非表示にしてシステムトレイにアイコンを入れたり，その反対をする。ウィンドウを非表示にすれば，タスクバーには表示されなくなるからだ。
しかし，タスクバーに表示されなければ，いったん最小化したあとに直接的に元に戻すことができない。そこでシステムトレイのアイコンをクリックしたときに，なんかのメニューを表示することにしよう。内容はシステムメニューと同じでいいが，ポップアップメニューとして別に作ってやる必要がある。最後に，そのポップアップメニューからのメッセージをハンドルするメッセージ関数を作る。流れとしてはこんな感じだ。
システムトレイにアイコンを入れるには，次の関数を使用する。

```
BOOL WINAPI Shell_NotifyIcon( DWORD dwMessage,
                               PNOTIFYICONDATA pnid );
```

第1引数は操作方法を指定し，NIM_ADDで追加，NIM_DELETEで削除，NIM_MODIFYだと変更ということになる。第2引数は，アイコンなどの情報が入った構造体へのポインタだ。この構造体は毎回使い回しができるので，先にCMainFrameのメンバとして作成し，構造体のメンバを埋めておこう。これもLoadFrame()でやればいい。

```
m_pNotifyIconData = new NOTIFYICONDATA;
m_pNotifyIconData->cbSize = sizeof(NOTIFYICONDATA);
m_pNotifyIconData->hWnd = m_hWnd;
m_pNotifyIconData->uID = 0;
m_pNotifyIconData->uFlags = NIF_ICON|NIF_MESSAGE|NIF_TIP;
m_pNotifyIconData->uCallbackMessage = WM_SYSTRAYCLICKED;
m_pNotifyIconData->hIcon = LoadIcon( AfxGetInstanceHandle(),
 MAKEINTRESOURCE(IDR_MAINFRAME) );
strcpy( m_pNotifyIconData->szTip, "wchange" );
```

cbSizeにはこの構造体のサイズが入る。正当性のチェックだと思えばいい。hWndは，システムトレイのアイコン上でマウス操作が行われた場合，メッセージが送られてくるウィンドウのハンドルを指定する。uIDはアプリ

ケーションごとの(システムトレイに入れる)アイコンのIDだ。メッセージと一緒にこのIDも送られてくるので，複数のアイコンをシステムトレイに入れたときには，このIDでどのアイコンが操作されたかを判別できる。

uFlagsは後回しにして，uCallBackMessageはhWndに送られるメッセージを示す。ここでは，セクション1でもやったとおり，WM_SYSTRAY CLICKEDをユーザーメッセージとして新たに作った。hIconはシステムトレイに表示するアイコンハンドルで，ここではリソースから読み込んでいる。LoadIcon()の引数はアプリケーションのインスタンスハンドルと，リソースの名前だ。インスタンスというのは……う〜ん，なんと説明してよいのか(筆者もよくは理解していない。申し訳ない)，クラスのインスタンスとかと同じで，アプリケーションの「実体」とかそんな感じじゃないかと思うんだが，今の場合はアプリケーションを識別するハンドルとでも思っておいていいんじゃないだろうか。とりあえず，「インスタンスハンドルをよこせ」といわれたら，黙ってAfxGetInstanceHandle()の引数を渡しておけばいい。リソースの名前は，リソースIDをMAKEINTRESOURCE()マクロに喰わせてやればいい。この辺は「構文」みたいなもんだ。

最後のszTipは64バイトの配列で，マウスカーソルをアイコンの上に持っていったときに表示されるチップヘルプの文字列だ。さて，先ほど説明をスキップしたuFlagsだが，アイコン，メッセージ，チップのうち，どれを有効にするかを示すフラグである。つまり，たとえばメッセージを無効にすればメッセージは送られてこないし，チップを無効にすればチップは表示されない(アイコンを無効にするとどうなるんだろう？)。全部有効にしたいなら，上のようにすべてのフラグを論理和でつなげばよい。

実際にこれらの構造体が使われるのは，ウィンドウが最小化，あるいはそこから元に戻されたときである。そのときに送られてくるメッセージは，WM_SIZEだ。これをハンドルしたら，次のようにすればいい。

```
void CMainFrame::OnSize(UINT nType, int cx, int cy)
{
  CFrameWnd::OnSize(nType, cx, cy);
  if( m_pNotifyIconData )
  switch( nType ){
  case SIZE_MINIMIZED:
    ShowWindow( SW_HIDE );
    Shell_NotifyIcon( NIM_ADD, m_pNotifyIconData );
    break;
  case SIZE_RESTORED:
    ShowWindow( SW_SHOW );
    Shell_NotifyIcon( NIM_DELETE, m_pNotifyIconData );
    break;
  }
}
```

nTypeは最小化されたときSIZE_MINIMIZEDを，元に戻されたときはSIZE_RESTOREDとなる。SIZE_RESTOREDは最大化から戻されたときにも渡されるが，いまのウィンドウで最大化はありえない。念のため，m_pNotifyIconDataが作られていないうちは無視するようにしてある。アイコンを表示したら，今度はそのアイコンから送られてくるメッセージをハンドルしよう。勘違いしないでもらいたいのは，あくまで「アイコン上でマウス操作したときに送られるメッセージ」であって，そのあとに表示されるメニューから送られてくるメッセージではない。もちろんそちらもハンドルしなければならないのだが，それはメニューを作ったあとの話だ。つまり，このあとはメッセージが目白押しってことだな。ということで，さっきのWM_SYSTRAYCLICKEDユーザーメッセージを処理するわけだが，もちろんClassWizardは使えないので，手動で書き込む。セクション1でやったとおり，ON_MESSAGE()を使えばよいのだが，もうくどくど説明はしないぞ。とにかく，

```
LRESULT CMainFrame::OnSystrayClicked( WPARAM wParam,
LPARAM lParam )
```

というメッセージ関数を作ろう。さてここでだ，実際にどういった操作が行われたときに，どんな値がパラメータとして渡されるのだろう。筆者が見た限り，その辺のことはヘルプには書いていなかったので，ちょっと実際に動かして操作してみた。

```
wParam:NOTIFYICONDATAで渡されたuID
lParam:下位8ビット
 1:左ボタンダウン
 2:左ボタンアップ
 3:左ボタンダブルクリック
 4:右ボタンダウン
 5:右ボタンアップ
 6:右ボタンダブルクリック
```

といったところだ。とりあえずlParamの下位8ビットが0でなければ，メニューを表示していいだろう。おっとっと，その前にポップアップメニューを作らなきゃ。これも使い回しするので，CMainFrameのメンバとしてあらかじめ作っておこう。やっぱりLoadFrame()ででもやっておこうか。メニューの作成に親ウィンドウのハンドル等は必要ないので，別にコンストラクタでも構わないんだけど。ヘッダで，

```
CMenu m_Menu;
```

としてメンバとしたら，

```
m_Menu.CreatePopupMenu();
m_Menu.AppendMenu( MF_STRING, SC_RESTORE,
                        "元のサイズに戻す(&R)" );
m_Menu.AppendMenu( MF_SEPARATOR );
m_Menu.AppendMenu( MF_STRING, SC_CLOSE,
                                "閉じる(&C)" );
m_Menu.AppendMenu( MF_SEPARATOR );
m_Menu.AppendMenu( MF_STRING, ID_APP_ABOUT,
                   "バージョン情報(wchange) (&A)..." );
```

くらいでいいか。SC_RESTOREとSC_CLOSEというのはシステムメニューのIDだが，ほかのコマンドIDとぶつかることはないので，このまま使ってしまった。で，OnSystrayClicked()では，

```
if( lParam&0xff ){
  POINT point;
  GetCursorPos( &point );
  m_Menu.TrackPopupMenu
                ( TPM_LEFTALIGN|TPM_LEFTBUTTON,
    point.x, point.y, this, NULL );
}
return 0;
```

としておけばいいだろう。マウスカーソルの座標はメッセージパラメータでは渡されないので，まずGetCursorPos()で取得する。TrackPopupMenu()がメニューを表示するメソッドで，第1引数はマウスの座標(というか，次の引数で渡される座標)がメニューのどの位置にくるか，および左右どちらのボタンでメニューの選択ができるかを示すフラグだ。TPM_LEFTALIGNはメニューの左，つまりマウスの右側に表示されることを示し，TPM_LEFTBUTTONは左ボタンでメニューの選択ができることを示している。

次の2つはいいとして，第4引数は，このメニューのメッセージを受けるウィンドウのポインタだ。最後は，このポップアップメニューが消去されずにクリックできる領域をRECT構造体のポインタで渡す。なにをいっているのかわからないかもしれないが，NULLにしておけば，メニューの外をク

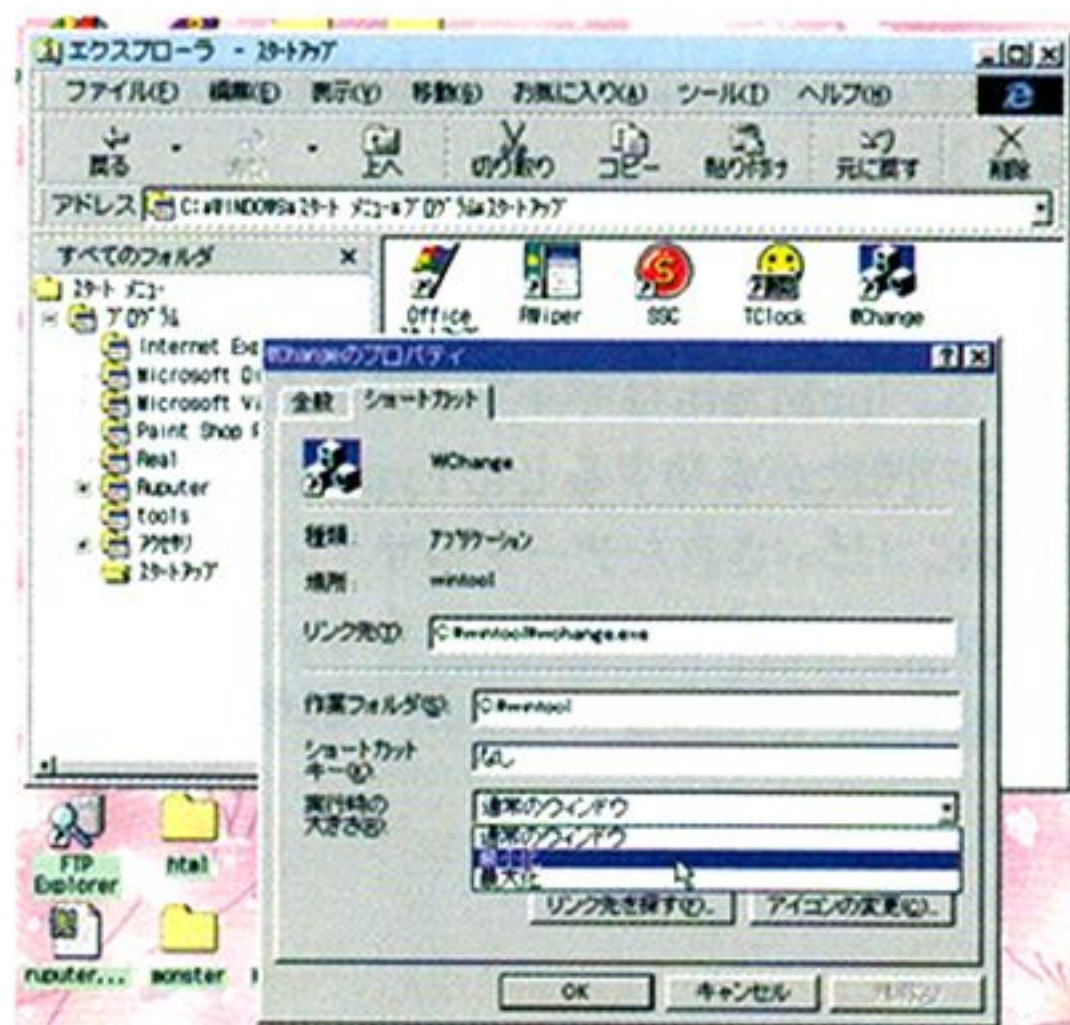
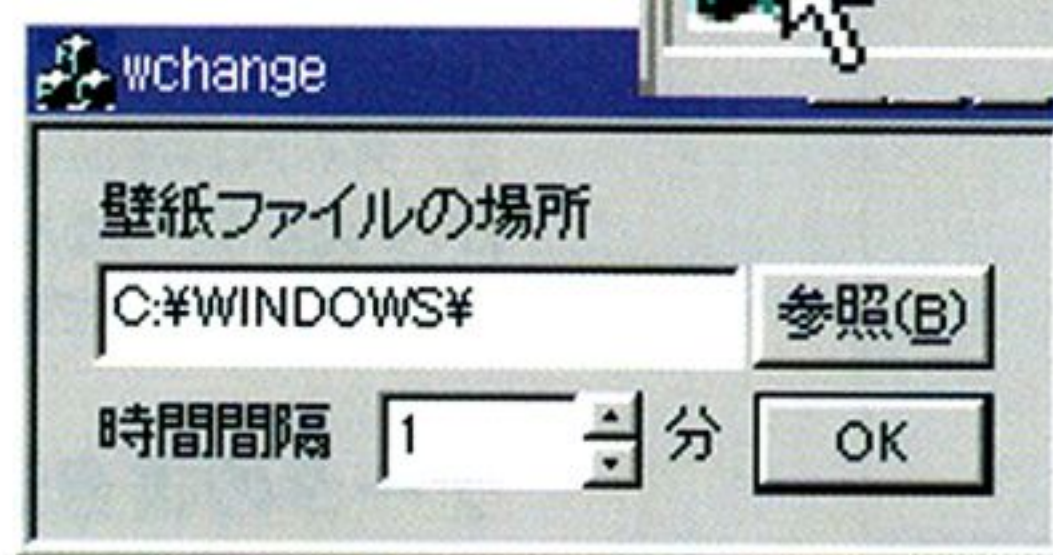



図9　最小化

図10　wchange-2，wchange-3

図11　soundprop

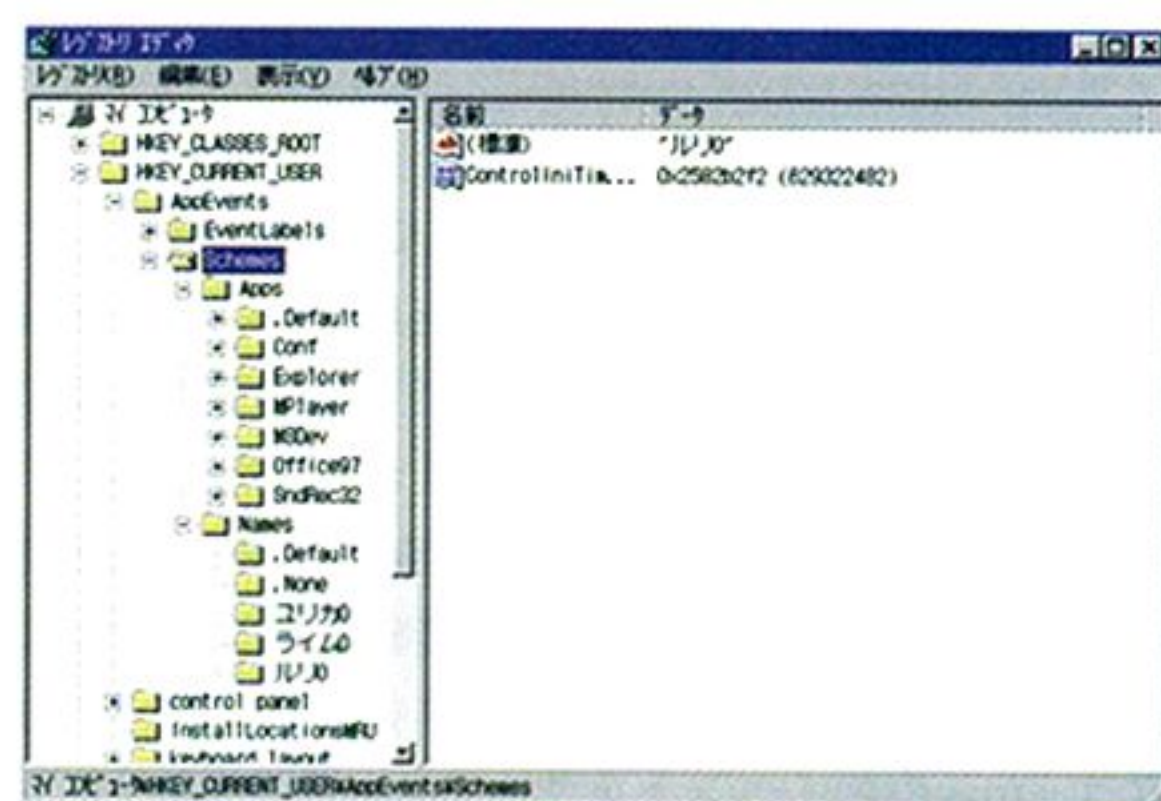
図12　regedit-2

リックすればメニューは消えてくれるようになる。それが普通だろう。

これで第2段階も終了。最後はこのメニューからのメッセージを取る。このメニューからのコマンドは，WM_COMMANDで送られる。と，いうことは，だ。バージョン情報のID_APP_ABOUTはすでにハンドルしてあるので，残りのSC_RESTOREとSC_CLOSEを取ったときに，今度はさっきと反対にWM_SYSCOMMANDで送り直してやればいいことになる。やっぱりClassWizardでは無理なので，手動でOnScRestore()とOnScClose()メッセージ関数を作ってやろう（OnClose()というのはすでにほかのメッセージ関数として登録されているので，使わないほうがよい）。

```
void CMainFrame::OnScRestore()
{
  PostMessage( WM_SYSCOMMAND, SC_RESTORE );
}

void CMainFrame::OnScClose()
{
  Shell_NotifyIcon( NIM_DELETE, m_pNotifyIconData );
  PostMessage( WM_SYSCOMMAND, SC_CLOSE );
}
```

ここで，今度はメッセージを送る関数として，SendMessage()ではなくPostMessage()を使ってみた。この2つのメッセージの違いは，SendMessage()がそのメッセージをウィンドウが処理するまで戻ってこないのに対して，PostMessage()はメッセージキューにメッセージをポストして，すぐに処理を返すという点だ。つまり，PostMessage()はとりあえず残りの処理を先に済ませてしまってから，あとでそのメッセージを処理するという形になる。デメリットとしては，メッセージハンドラからの戻り値を受けることができないという点だが，今は単にイベントが起こったことを知らせたいだけなので，問題はない。無駄に処理を深くするのはあまりいい気分ではないので，戻り値が必要なくて，急を要さない場合はPostMessage()のほうがスマートだ。

あと，起動したら最初から最小化してほしいって要望もあるだろうけど，それはWindowsのスタートメニューのほうで設定できる。タスクバーのプロパティ，スタートメニューの追加で登録したら，今度は詳細ボタンを押して，いま追加したショートカットを探す。そのアイコンで右クリックし，そこからプロパティを選択すると，ダイアログが開くから，ショートカットタブの中の［実行時の大きさ］を［最小化］にする（図9）。

これで，起動したらいきなり最小化されるはず，だったんだけど，文字どおり最小化されるだけで，タスクバーに表示されたまんまで，システムトレイにアイコンは表示されない。実行時に最小化すると，WM_SIZEでSIZE_MINIMIZEDが送られてこないんじゃないかと疑ったけど，調べてみるとちゃんときている。よくわからんけど，BOOL型のメンバ変数を用意してシステムトレイにアイコンを表示しているかのフラグとし，WM_SIZEのSIZE_MINIMIZEDはまだシステムトレイにアイコンを表示していないとき，SIZE_RESTOREDはすでにシステムトレイにアイコンが表示されているときだけ処理するようにしたら，なんか知らんけどちゃんと動いた。ま，Windowsだし，なにがあっても不思議じゃないか。

そうそう，あとCWchangeApp::InitInstance()の中の最後のほうの，

```
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
```

もコメントアウトしておく。実行時に最小化しちゃうと，ここにくるまでにすでにウィンドウを非表示にして，システムトレイにアイコンが入ってしまっているからだ。消してしまうと，普通に起動した場合に困るんじゃないかと思うかもしれないが，別に困らなかった（おい）。ま，Windowsだし。

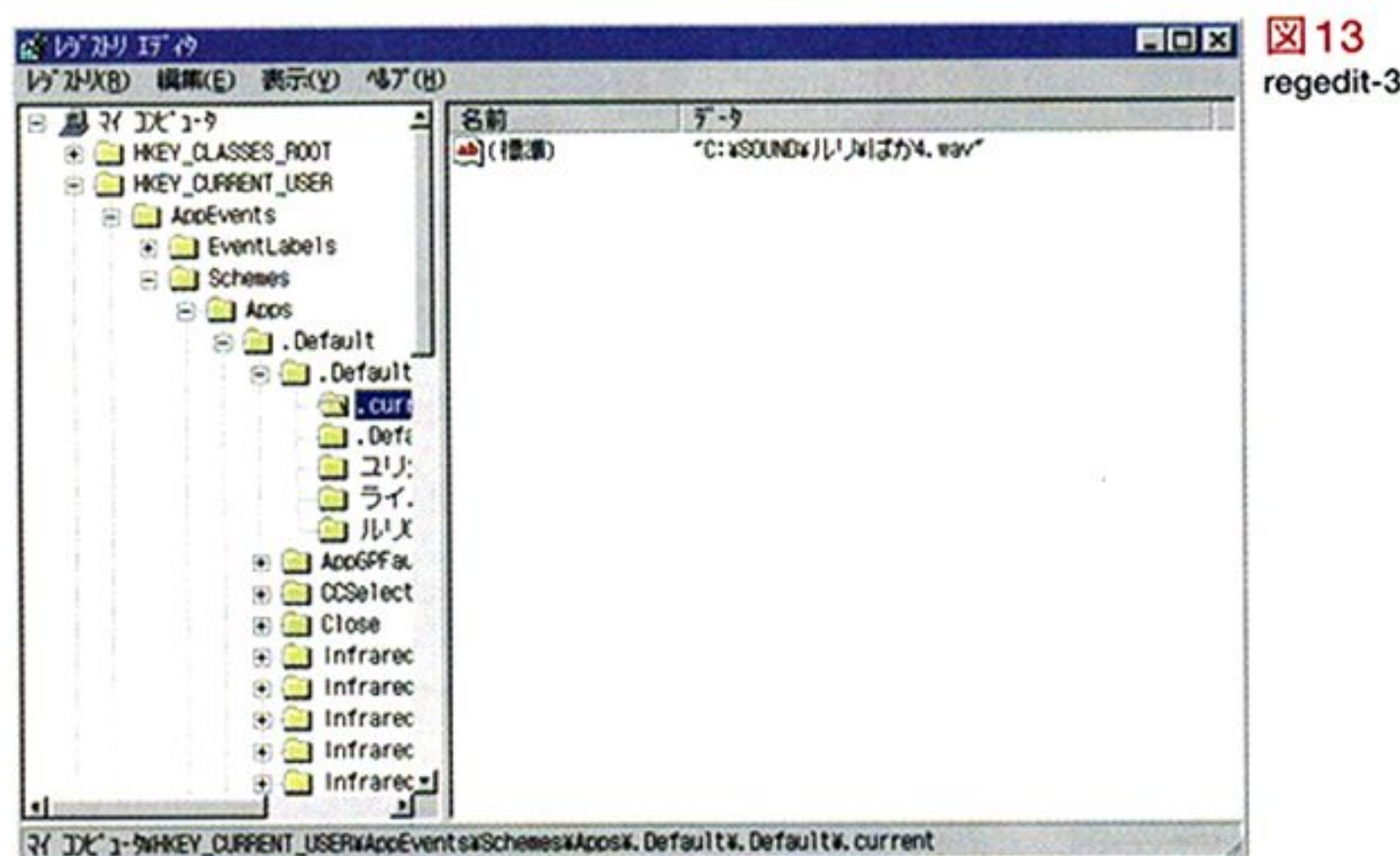
これでぐぐっと完成度が上がったよね（サンプルwchange2）。ウィンドウサイズもぴったり，メニューもなくなってシンプルになった。最小化したら，ちゃんとシステムトレイに入るのも確認してほしい（図10）。

サウンドチェンジャ機能

へい，お待ち。やっとサウンドのほうに入れるよ。その前に，まずサウンドを登録しとくれ。コントロールパネルのサウンドのプロパティで，イベントごとにサウンドファイルを指定したら，下のほうの［登録］ボタンを押して，登録名を適当につけるだけだ（図11）。ひとつじゃチェンジャの意味がないからね。サウンドをかき集めて，片っ端から登録しちまえ。このとき，サウンドファイルを指定するのに，［参照］ボタンでいちいちダイアログから選択するのは面倒だから，エクスプローラからドロップしちまうのが手っ取り早い。

さて，ここで設定したサウンドがどこ行っちまうかというと，いうまでもなくレジストリだ。レジストリエディタでHKEY_CURRENT_USER¥AppEvents¥Schemesを開いてみればすぐわかる。このなかに，AppsとNamesというキーがあるだろう（図12）。Namesのほうは，その下に設定したサウンドの個数だけキーがあり，その中に単に設定の名前が保存されているだけだ（キー名の最後に0がついているのが意味不明だが）。このキーのうち，.Defaultが「標準のサウンド設定」，.Noneが「サウンドなし」に当たる。

ちなみにレジストリ内では，設定の名前ではなくキー名で管理される。Schemesのキーにカーソルを合わせたときに右側のウィンドウに表示される，（標準）に書き込まれたデータは，現在選択されているサウンド設定のキー名だ。ではAppsはというと，この下にアプリケーションごとのキーが，さらに下にイベントごとのキーがあり，その中に各設定，および現在の設定

図13
regedit-3

のキー名でファイル名が保存されている(図13)。

この図の場合，Apps直下の.DefaultはWindows標準のイベントであることを示し，その下の.Defaultは「一般の警告音」であることを示している。この中の.currentに入っているファイルが，現在「一般の警告音」に設定されているファイルというわけだ(.Noneがない理由はいうまでもない)。だから，たとえば「ルリ」という設定に変更したい場合は，まずNamesの中のキー内を検索し，設定名「ルリ」に対するキーが「ルリ0」であることを調べ，今度はApps内のイベントをサーチし，キー「ルリ0」に入っているファイル名をキー.currentにコピーすればいい。あと，Schemesキーの(標準)も「ルリ0」にする。ところで，それぞれのイベントキーがどのイベントに対応しているかは，Schemesの上のEventLabelsの中身を見ればわかるはずだ。

では上で説明した手順で，プログラミングしていこう。壁紙の変更と同じタイミングでサウンドも変更するということだったので，記述するのはOnTimer()の中でいいだろう。まず必要となるレジストリ操作関数を挙げていこう。

●キーのオープン

```
LONG RegOpenKeyEx( HKEY hKey, LPCTSTR lpSubKey,
DWORD ulOptions,
 REGSAM samDesired, PHKEY phkResult );
```

キー内のデータの操作は，キーハンドル(HKEY)で行うので，まずはそのキーハンドルを取得する必要がある。このRegOpenKeyEx()は，特定のキーハンドルからそのサブキーのハンドルを取得する関数だ。ではいちばん初めのキーハンドルはというと，HKEY_CURRENT_USERといったハンドルを指定する。したがって，深い階層のキーハンドルを取得しようと思ったら，この関数を幾重にも入れ子にする必要がある……と思ったら大間違い。ヘルプには特に書いていないのだが，lpSubKeyには"AppEvents¥¥Schemes¥¥Names"などとして，¥マークで区切って深い階層を直接指定できる(クオーテーション内で¥マークを重ねるのは，エスケープシーケンスに引っかかるからってのは，いまさらいう必要もないと思うが)。第三引数のulOptionsはリザーブで0を指定しなければならない。次のsamDesiredはセキュリティのためのアクセスマスクを示すが，読み込むときにはKEY_READ，書き込むときにはKEY_WRITEと覚えておけばいいだろう。最後phkResultは取得するキーハンドルへのポインタだ。

キーのクローズ

```
LONG RegCloseKey( HKEY hKey );
```

オープンがあるからにはクローズがある。説明の必要はないわな。

データの読み出し

```
LONG RegQueryValueEx( HKEY hKey, LPTSTR lpValueName,
 LPDWORD lpReserved, LPDWORD lpType, LPBYTE lpData,
LPDWORD lpcb      Data );
```

hKeyで指定したキーの中のlpValueNameという項目のデータを，lpDataが示すバッファへコピーする。lpcbDataはバッファのサイズの入ったWORD変数へのポインタで，この関数が成功するとlpTypeが示す変数にはタイプが，lpcbDataには実際にコピーされたデータのサイズが入れられる。タイプは数値がREG_DWORD，文字列がREG_SZくらい覚えておけばいいだろう。ちなみにlpValueNameをNULLにすると，(標準)という項目のデータを取得できる。

データの書き込み

```
LONG RegSetValueEx( HKEY hKey, LPCTSTR lpValueName,
 DWORD Reserved, DWORD dwType, CONST BYTE *lpData,
                                     DWORD cbData );
```

hKeyで指定したキーの中にlpValueNameという名前で，dwTypeタイプのlpDataが示すデータをcbDataバイト書き込む。RegQueryValueEx()同様，lpValueNameをNULLにすると，(標準)という項目に書き込める。

キーの情報取得

```
LONG RegQueryInfoKey ( HKEY hKey, LPTSTR lpClass,
                                   LPDWORD lpcbClass,
 LPDWORD lpReserved, LPDWORD lpcSubKeys, LPDWORD
                                lpcbMaxSubKeyLen,
 LPDWORD lpcbMaxClassLen, LPDWORD lpcValues,
                        LPDWORD lpcbMaxValueNameLen,
 LPDWORD lpcbMaxValueLen, LPDWORD
                           lpcbSecurityDescriptor,
 PFILETIME lpftLastWriteTime );
```

hKeyで指定したキーのさまざまな情報を取得できるが，今回使うのはこれらの引数のうち，サブキーの数を取得するlpcSubKeysと，サブキーの最大長を取得するlpcbMaxSubKeyLenだけ。必要ないものはNULLにしておけばよい。

サブキーの列挙

```
LONG RegEnumKeyEx( HKEY hKey, DWORD dwIndex,
                                       LPTSTR lpName,
 LPDWORD lpcbName, LPDWORD lpReserved, LPTSTR lpClass,
 LPDWORD lpcbClass, PFILETIME lpftLastWriteTime );
```

hKeyで指定したキー内の0から始まるdwIndex番目のサブキー名を，lpNameで示すバッファにlpcbNameが示すバイト分コピーする。関数が正常に終了すると，lpcbNameにはコピーされたサイズが格納される。クラス名や更新時間も取得できるが，今回は必要ないのでNULLにしておく。

この6つの関数があれば，こと足りるはずだ。まずは設定数を取得してみよう。

```
HKEY hKey;
LPTSTR lpName;
DWORD SubKeys=0, cbName;
if( RegOpenKeyEx( HKEY_CURRENT_USER,
                      "AppEvents¥¥Schemes¥¥Names",
 0, KEY_READ, &hKey )==ERROR_SUCCESS ){
  if( RegQueryInfoKey ( hKey, NULL, NULL, NULL, &SubKeys,
                                          &cbName,
```

```
  NULL, NULL, NULL, NULL, NULL, NULL )==ERROR_
                                         SUCCESS ){
   SubKeys -= 2;
  }
  if( SubKeys<2 ){ RegCloseKey( hKey ); return; }
```

SubKeysから2を引いているのは，「標準の設定」と「サウンドなし」の分だ。これを差し引いて，2個以上の設定がない場合は，サウンドチェンジャは意味を成さないのでキーを閉じて終了する。続いて，ランダムに設定のキーを選択する。

```
  SubKeys = rand()%SubKeys;
  lpName = new STR[++cbName];
  RegEnumKeyEx( hKey, SubKeys+2, lpName, &cbName,
                            NULL, NULL, NULL, NULL );
  RegCloseKey( hKey );
 } else return;
```

cbNameを加算しているのは，終端のnull文字の分だ。SubKeysに2を加えているのはいうまでもなく「標準の設定」と「サウンドなし」をスキップするため(.Defaultと.Noneはおそらく先頭にあるはずだ)。

```
 if( RegOpenKeyEx( HKEY_CURRENT_USER,
                               "AppEvents¥¥Schemes",
  0, KEY_WRITE, &hKey )==ERROR_SUCCESS ){
   RegSetValueEx( hKey, NULL, NULL, REG_SZ,
                         (LPBYTE)lpName, cbName+1 );
   RegCloseKey( hKey );
 }
```

これで，Schemesの(標準)にいまから変更する設定のキー名が書き込まれたはずだ。cbNameにまた1を足しているのは，前のRegEnumKeyEx()でnullを含まない文字数を格納されてしまったから。ここから先はもう力技だろう。

```
 if( RegOpenKeyEx( HKEY_CURRENT_USER,
                         "AppEvents¥¥Schemes¥¥Apps",
  0, KEY_READ, &hKey )==ERROR_SUCCESS ){
   HKEY hKey2;
   int index = 0;
   LPTSTR lpApp = new char[256];
   LPTSTR lpEvent = new char[256];
   LPTSTR lpData = new char[256];
   DWORD cbApp, cbEvent, cbData;
   while( RegEnumKeyEx( hKey, index++, lpApp,
                              &(cbApp=256), NULL,
    NULL, NULL, NULL )!=ERROR_NO_MORE_ITEMS ){
     if( RegOpenKeyEx( hKey, lpApp, 0, KEY_READ, &hKey2 )
      ==ERROR_SUCCESS ){
       HKEY hKey3;
       int index2 = 0;
       while( RegEnumKeyEx( hKey2, index2++, lpEvent,
                              &(cbEvent=2    56), NULL,
        NULL, NULL, NULL )!=ERROR_NO_MORE_ITEMS ){
         if( RegOpenKeyEx( hKey2, lpEvent, 0, KEY_READ,
                                           &hKey3 )
          ==ERROR_SUCCESS ){
           HKEY hKey4;
           if( RegOpenKeyEx( hKey3, lpName, 0, KEY_READ,
                                           &hKey4 )
            ==ERROR_SUCCESS ){
             DWORD type;
             RegQueryValueEx( hKey4, NULL, NULL, &type,
                                        (LPBYTE)lpData,
              &(cbData=256) );
             RegCloseKey( hKey4 );
             if( RegOpenKeyEx( hKey3, ".current", 0, KEY_WRITE,
                                               &hKey4 )
              ==ERROR_SUCCESS ){
               RegSetValueEx( hKey4, NULL, NULL, REG_SZ,
                                        (LPBYTE)lpData,
                cbData+1 );
               RegCloseKey( hKey4 );
             }
           }
           RegCloseKey( hKey3 );
         }
       }
       RegCloseKey( hKey2 );
     }
   }
   RegCloseKey( hKey );
   delete []lpApp;
   delete []lpEvent;
   delete []lpData;
 }
 delete []lpName;
```

ぎょえええぇぇ～～っ！もう堪忍して～っ！なんて弱音を吐いてる場合じゃないぞ。まあ，RegEnumKeyEx()をwhileループで2階層丸々検索してるからなぁ。がんばって解析してくれ。多分間違いはないと思うけど。赤ペンでも持って，「このifとこの閉じカッコが対応して……」なんてやると，少しはわかりやすいかもしれないぞ。キーやデータのサイズは面倒なんで最大256バイト決め打ちしちゃったけど，普通足りるでしょ。

う～ん，がんばったおかげで，なんとなんと，もうできちゃったよ。やっぱウィンドウとかメッセージ周りが必要ないとこは，説明が楽でいいなぁ(勢いで押せるから)。さてさて，ミスがないことを祈りつつ，神にもすがる気持ちでビルドしてみよう。なんせレジストリだからね。あの辺をいじってる分には，Windowsが死ぬってほどのことはないだろうけど，やっぱ不安だよね。さて，深呼吸をして，気を落ち着けて。マシンのバックアップは取ってあるね。じゃ，意を決して実行！

おお～っ！ルリからユリカになったぁ～！(バカ)

明けましておめでとうございます

おいおい，いまごろなにいってんだよ。まあ確かに原稿執筆時点では今日は1月2日だけど。でも年末年始を押してがんばったおかげで，割と実用的なサンプルになったんじゃないかと思う。レジストリの扱い方とかも覚えたよね。本当はこういうものはMFCを使わないで作りたいところなんだけど。サイズは無駄にでかいし，ドキュメントクラスはまったくいじらなかったし。そういう記事だからしょうがないか(といいつつも，筆者のホームページにはMFCを使っていないほぼ同じものがアップされている)。次回の予告はセクション1のほうでやっちゃったから，こっちは募集でもしてみようか。こんなツールがほしいとか，こんなのはどうやんの？　とかいうことがあれば，愛読者ハガキの裏でもいいから書いて送ってくれ。筆者の能力の範囲内ならば，この記事内で紹介できるかもしれない。能力を超えちゃったら，「特命リサーチ200X」のインターネットセクションにでも流しちゃうか。念のためにいっとくと，筆者はこの番組が大嫌いだ。

Tcl/Tkによるパズルゲームと解法

広井誠　Hiroi Makoto

春号の解説に続いて，夏号でも広井氏によるTcl/Tkの投稿プログラムを紹介する。題材はプログラムの入門にはぴったりのパズルゲームだ。Tcl/Tkによるパズルプログラムを２本とＣ言語による解法プログラムをお届けしよう。

パズルゲーム「TEN(テン)」

「TEN(テン)」は，足して10になる2枚のカードを取り除いていくパズルゲームです。カードは赤，青，黄，緑の4種類があり，それぞれ1から9までのカードが4枚ずつ，計144枚のカードをすべて取り除けばクリアです。

このゲームは，トランプのひとり遊び「TEN」が元ネタです。最初はTcl/Tk入門講座で使用するサンプルプログラムとして作り始めたのですが，ほかのゲームからアイデアを拝借してみると，数字を表示するだけのシンプルな画面ながら，それなりに遊べるようです。オリジナリティのカケラもありませんが，暇つぶしや気分転換に遊んでみてください。

ルール

TENには2種類のゲームが用意されています。

(A)上海モード

カードは6行6列4段に積まれていて，次の条件を満たす2つのカードを取り除くことができます。

(1) 同じ色で足して10になること
(2) 4辺の中でほかのカードと接していない辺があること

(2) の条件により，最初から中央にあるカードは取ることができません。

(B)四川省モード

カードは9行18列に配置され，次の条件を満たす2つのカードを取り除くことができます。

(1) 同じ色で足して10になること
(2) カードがない場所を通って，3本以下の連続した直線でカードが結べること

上下左右の隣にあるカードは，(1)の条件を満たせば取り除くことができます。(2)のルールは，小野晋二氏が作成されたパズルゲーム「四川省」から拝借しました。四川省は同じ種類の麻雀牌を取り除くゲームです。素晴らしいゲームをフリーウェアとして公開された小野晋二氏に感謝いたします。

2つのゲームともに，スコアはクリアするまでの時間です。トップテン入りを目指して，がんばってください。また，カードはランダムに配置されるため，クリアできる保証はありません。ご注意くださいませ。

実行方法

ゲームを実行するには，まずTcl/Tkをインストールしてください。私が使ったバージョンはTcl/Tk8.1a2です。Windows 版はtcl812a.exeを使います。インストーラを実行するとインストーラが起動され，「スタート」メニューの「プログラム」フォルダにTcl/Tkのフォルダが作成されます。あとは，拡張子がtclのファイル（Tclスクリプトファイル）をダブルクリックすれば，ゲームが実行されます。

そのほかのバージョンでゲームを起動したときにエラーが発生するようなら，プログラム中で使用されているメッセージなどを英文にしてみてください。CD-ROMに収録されているファイルは以下のとおりです。

- TEN_上海.TCL
  - 上海モード用Tclスクリプトファイル
- TEN_四川.TCL
  - 四川省モード用Tclスクリプトファイル
- TEN_HELP.TXT
  - 取扱説明書
- TEN_SUB.TCL
  - 共通サブルーチン

TEN_SUB.TCLは，2つのゲームで共通に使うサブルーチンを集めたスクリプトファイルなので，ダブルクリックしてもゲームは実行できません。また，スコアを記録するため，ハードディスクなどの書き込み可能なメディア上で実行してください。

カードゲームを作る場合，カードを重ね合わせるときの処理が面倒です。カードの表示だけではなく，マウスでクリックしたときの判定では，上にあるカードを選択しなければいけません。Tcl/Tkを使うと，このような面倒な処理はすべてシステムが行ってくれるため，簡単にプログラムを作ることができます。また，適当なカードデータを用意すれば，もっと見栄えのするゲームになるでしょう。ゲームに飽きたら，いろいろと改造して遊んでみてください。

ナンバープレース

続いて，お馴染みの方も多いと思われるパズルゲーム「ナンバープレース」をTcl/Tkで作成しました。縦，横，太線の枠の中に，異なる数字をひとつずつ埋め込んでいきます。たとえば，盤面が9行9列の場合，縦に1から9ま

図1　これが上海モード

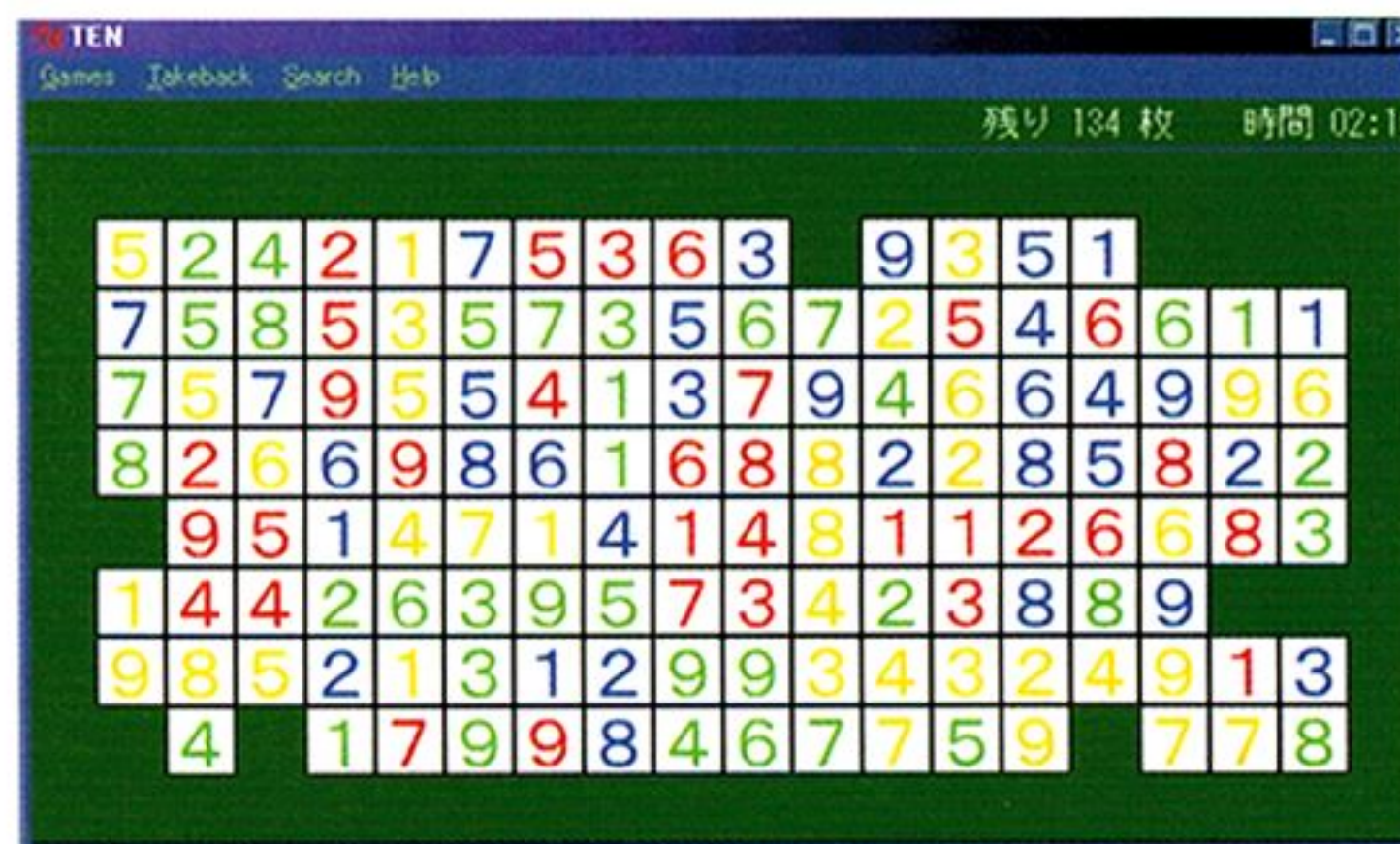



図2　こちらは四川省モードのゲーム画面

図3　ナンバープレース。
補助用のウィンドウも表示される

での数字をひとつずつ，横に1から9までの数字をひとつずつ，太線の枠（3行3列）の中に1から9までの数字をひとつずつ入れます。盤面にはヒントの数字が表示されているので，これを頼りに空いている場所に数字を入れてください。盤面は9行9列のほかに，12行12列（1から12まで）と16行16列（1から16まで）を用意しました。

nmpl09.tcl
　9行9列盤用Tclスクリプトファイル
nmpl09.dat
　データファイル1
nmpl0901.dat
　データファイル2
nmpl12.tcl
　12行12列盤用Tclスクリプトファイル
nmpl12.dat
　データファイル1
nmpl1201.dat
　データファイル2
nmpl16.tcl
　16行16列盤用Tclスクリプトファイル
nmpl16.dat
　データファイル1
nmpl1601.dat
　データファイル2
numplace.tcl
　共通のTclスクリプトファイル
nmplhelp.txt
　取扱説明書

問題の自動生成

このゲームに付属している問題はプログラムによって自動生成されたものです。問題を格納したデータファイルは，付属のプログラムによって作ることができます。CD-ROMに収録されたプログラムの実行ファイルはDOS用で，コマンドライン上で実行します。

9行9列盤の場合
　C> mknmpl09 /N10 > nmpl09＊.dat
　12行12列盤の場合
　C> mknmpl12 /N10 > nmpl12＊.dat
16行16列盤の場合
　C> mknmpl16 /N10 > nmpl16＊.dat

このプログラムは作成した問題を標準出力へ書き出すので，ファイルへリダイレクトしてください。ファイル名の先頭6文字は，データファイルの検索に使用するため変更してはいけません。＊にはいままでのデータファイルと重複しない文字列をセットしてください。/Nの後ろには，作成する問題数を指定します。16行16列盤の場合，問題の作成には時間がかかります。問題数は少な目に指定しましょう。

プログラムはC言語で作成しました。バッチファイルmakenmpl.batを実行すると，ひとつのソースファイル（makenmpl.c）から3つの実行ファイルを作成します。やっていることは簡単で，コンパイラの実行時にオプションでマクロを定義しているだけです。使用したコンパイラはVC++なので，ほかのコンパイラを使う場合は，オプションの設定を修正してください。

自動生成の基本的な考え方は，佐々木崇氏がX-BASICで作成されたプログラム［1］とほぼ同じです。

　(1) 乱数を使って数字を配置する
　(2) 乱数を使ってヒントの数字を選ぶ
　(3) 解けるようになるまで (2) を繰り返す

(1)では，数字を矛盾なく配置するため，バックトラックを使っています。今回は再帰を使ってバックトラックを実現してますが，256回の再帰呼び出しを行うmknmpl16.exeでも，システムスタックは16Kバイトあれば動作します。

一般に，再帰呼び出しは初心者には難しいテクニックと思われていたり，上級者でも，関数呼び出しのオーバーヘッドを理由に再帰を敬遠している方もいるようです。

ところが再帰呼び出しは，思われているよりもずっと簡単で効率的なのです。バックトラックのような複雑なアルゴリズムでも，再帰呼び出しを使えば簡単にプログラムを作ることができます。プログラミングに興味をお持ちの方は，ぜひ再帰プログラミングにも挑戦してください。

作成された問題は必ず解くことができますが，乱数に頼っているので，面白さの点は無保証です。難易度についても考慮していないので，簡単すぎたり，難しすぎる問題があるかもしれません。工夫する余地は十分にあると思われるので，興味のある方はプログラムを改造してみてください。

また，データファイルの構成は単純なテキストファイルなので，エディタで問題を作ることもできます。データファイルは1行でひとつの問題を表します。左上隅のマスが1文字目に，その左隣が2文字目と順番に対応していき，右下隅のマスが最後の文字となります。したがって，9行9列盤では1行81文字，16行16列盤では1行256文字のテキストファイルとなります。

データは0から9の数字と，AからGまでのアルファベットを使っています。0が空白を表します。1から9までの数字はそのままで，10から16までの数字をAからGまでのアルファベットで表します。

その他

このプログラムはナンバープレースを解く，という基本的な機能しか実装していません。使っていると，自動解法や問題作成のためのエディット機能がほしいと思われる方も多いでしょう。そのときは，ぜひ自分で改造してみてください。

次項ではCによる解法プログラムを紹介しますが，バックトラックによる解法をとらなければTcl/Tkでも十分実用的な速度で処理できるでしょう。アルゴリズムは，中村隆生氏と舩本昇竜氏の記事[2][3]が参考になります。興味のある方は挑戦してみてください。

ナンバープレース解法プログラム

「ナンバープレース」の解法プログラムです。

拙作の「ナンバープレース」には，問題自動生成プログラムmknmpl??.exeが付属していますが，そこに組み込まれている解法アルゴリズムでは解けない問題があったので，バックトラックにより力ずくで解くことにしました。ところで，ドキュメントには「解法プログラムはTclでも簡単に作成でき，短時間で答えが出せる」と書きましたが，バックトラックで問題を解くとなると，話が違ってくるのです。大きな盤面になるとTclでは時間がかかるので，C言語でプログラムを作りました。

ファイル名

numpla09.exe
　9行9列盤用解法プログラム
NP0901.DAT

問題ファイル1（確定サーチで解ける）
NP0902.DAT
問題ファイル2（複数の解がある問題）
numpla12.exe
12行12列盤用解法プログラム
NP1201.DAT
問題ファイル1（確定サーチで解ける）
NP1202.DAT
問題ファイル2（複数の解がある問題）
numpla16.exe
16行16列盤用解法プログラム
NP1601.DAT
問題ファイル1（確定サーチで解ける）
NP1602.DAT
問題ファイル2（複数の解がある問題）

確定サーチとは，ヒントの数字から確定できる数字を探すアルゴリズムのことです。これはあとで詳しく説明しましょう。また，NP??02.DATのように複数の解がある問題は，当然のことですがパズルの問題としては失格です。バックトラックにより全解探索を行っていることを確かめるために用意しました。

プログラムの実行

プログラムはDOS窓で実行します。

```
9行9列盤の場合
  C> numpla09 問題ファイル名
12行12列盤の場合
  C> numpla12 問題ファイル名
16行16列盤の場合
  C> numpla12 問題ファイル名
```

問題ファイル名が省略されると，標準入力から問題を読み込みます。解答は標準出力へ書き出すので，ファイルに保存する場合はリダイレクトしてください。

問題ファイルの書式ですが，単純なテキストファイルです。0が空いている場所を表します。あとは数字をそのまま書き込みます。たとえば9行9列盤の場合は，1行に9個の数字を順番に並べます。ただし，数字と数字の間は空白で必ず区切ってください。#から始まる行はコメントとして扱われます。サンプルの問題ファイルを見てもらえば，すぐに理解できるでしょう。

プログラムはC言語で作成しました。バッチファイルmakenuma.batを実行すると，ひとつのソースファイル(numplace.c)から3つの実行ファイルを作成します。使用したコンパイラはVC++なので，ほかのコンパイラを使う場合は，オプションの設定を修正してください。

アルゴリズムについて

ナンバープレースは，ヒントの数字から次に示す条件を使って，空き場所の数字を決定することができます。

(1) 数字がひとつしか入らない場所を探す
(2) 縦，横，枠のそれぞれについて，置ける場所がひとつしかない数字を探す

これを「確定サーチ」と呼ぶことにします。空き場所に置くことができる数字は，その場所が属している，縦，横，枠のいずれにも使用されていない数字です。9行9列盤であれば，1から9までの数字の中から，縦，横，枠で使われている数字を削除すれば求めることができます。そして，残った数字がひとつであれば，その数字で確定することができます。これが条件(1)です。これで確定できない場合は条件(2)を使います。次の表を見てください。

表では3，5，6の数字が未確定です。ここで数字5に注目してください。縦の中で5を置くことができる場所は，3行目の1カ所しかありませんね。したがって，この場所は5に確定することができるのです。同じように，横，枠の中からもこの条件を満たす数字を探すことができます。

行	確定した数字	置くことができる数字
1	1	
2	0	[3, 6]
3	0	[3, 5, 6]
4	8	
5	4	
6	0	[3, 6]
7	2	
8	9	
9	7	

数字をひとつ確定すると，そのことにより条件(1)や(2)を満たす場所が出てくるので，その場所を探して数字を確定します。あとはこれを繰り返すだけです。この解き方は，人間がナンバープレースを解くときに使う方法です。拙作のナンバープレースは，入力できる数字を水色のボタンで表示しているので，条件(1)や(2)を満たす場所を簡単に探すことができるようになっています。雑誌に掲載されているナンバープレースは，この確定サーチだけでほとんどの問題が解けるのですが，世の中そう甘くはありません。これでは解けない難しい問題があるのです。

そのような難問をどうやって解いたらよいのでしょうか，筆者には見当もつかないのですが，コンピュータを使えば「総当たり」という力技で解を見つけることができます。つまり，確定サーチでも決まらない場所は，可能性のある数字を入れてみて，それで解けなければ違う数字を試してみる，という試行錯誤によって数字を決定しよう，というわけです。試行錯誤を実現するのに適したアルゴリズムがバックトラックです。パズルを解くプログラムでは常套手段であり，再帰を使えば簡単にプログラムを作ることができます。

プログラムについて

実際にプログラムを作る場合，置くことができる数字を高速で求めるための工夫が必要です。問題生成プログラムでは，場所ごとに置くことができる数字をフラグで表しています。数字が確定したら，縦，横，枠の各場所のフラグをクリアすればいいわけです。フラグが立っていれば，その数字を置くことができます。とてもわかりやすいデータ構造ですね。実際に，フラグをクリアするプログラムを示しましょう。

```
void clear_flag(int n, int x, int y)
{
  int i, j, x1, y1;
  int off = bitoff[n];
  /* 縦横のクリア */
  for(i = 0; i < SIZE; i++){
    flag[x][i] &= off;
    flag[i][y] &= off;
  }
  /* 枠のクリア */
  x1 = (x / GRX) * GRX;
  y1= (y / GRY) * GRY;
  for(i = 0; i < GRX; i++){
    for(j = 0; j < GRY; j++){
      flag[x1 + i][y1 + j] &= off;
    }
  }
}
```

配列boardが盤面を表します。置くことができる数字はビットで表し，配列flagに格納しています。配列bitoffはビットをクリアするためのデータを格納します。プログラムはフラグをクリアするだけの単純な処理なので，すぐに理解できるでしょう。

ところがバックトラックする場合，各場所ごとにフラグを持たせておくと都合が悪いのです。試行錯誤するのですから，数字を置くだけではなく取

り消す処理も必要です。このとき，フラグの状態も元に戻さなければいけないのですが，縦，横，枠の各場所に対して単純にフラグをセットするだけでは，元の状態に戻すことができないのです。次の表を見てください。

行	確定した数字	置くことができる数字
1	9	
2	0	[3，4，6]
3	0	[3，4，6]
4	5	
5	0	[1，6，8]
6	0	[1，6，8]
7	7	
8	0	[1，6，8]
9	2	

上の状態では，5，6，8行目に3を置くことができません。ここで，2行目の場所に3を置いてみましょう。すると，3行目で置くことができる数字は[4，6]になりますが，5，6，8行目のフラグに変化はありません。ここで，バックトラックが発生して，2行目の3を取り消すことになりました。3行目の位置は，フラグをセットすれば元の状態に戻りますが，5，6，8行目では無条件にフラグをセットすると，3を置くことができる状態になり矛盾してしまいますね。結局，フラグの状態を元に戻すには，盤上の数字からフラグを改めて作り直さなければいけないのです。これでは，大きな盤面になると時間がかかってしまいます。16行16列盤の問題生成に時間がかかるのは，これが原因だったのです。

この対策のために，フラグの状態を保存しておこうか，と考えたのですが，松田晋氏の記事[4]によい方法が書いてありました。それは，縦，横，枠のそれぞれについて，置くことができる数字をビットで保持しておく，という方法です。具体的に説明すると，縦，横，枠を表す3つの配列xflag[]，yflag[]，gflag[][]を用意し，まだ使っていない数字をビットで表します。このデータ構造にすると，場所(x，y)に数字を置く処理は，次のようにプログラムできます。

```
void write_number(int n, int x, int y)
{
  int off = bitoff[n];
  board[x][y] = 0;
  xflag[x] &= off;
  yflag[y] &= off;
  gflag[xgrp[x]][ygrp[y]]&= off;
}
```

配列xgrp[]とygrp[]はxとyから枠を求めるための配列です。3つの配列から数字を表すビットをクリアするだけです。その代わりに，場所(x，y)で置くことができる数字は，次の計算で求めなければいけません。

```
(xflag[x] & yflag[y]& gflag[xgrp[x]] [ygrp[y]])
```

いちいちAND演算しなければいけないので面倒なようですが，数字を取り消す処理はとても簡単になります。

```
void delete_number(int n, int x, int y)
{
  int on = biton[n];
  board[x][y]= n;
  xflag[x]|= on;
  yflag[y]|= on;
  gflag[xgrp[x]] [ygrp[y]]|= on;
}
```

配列 biton[]はフラグをセットするためのデータを格納しています。縦，横，枠の各フラグをセットするだけなので，短時間で元の状態に戻すことができます。バックトラックする場合，このデータ構造のほうが適しています。実際の探索プログラムは，次のようになります。

```
void search(int pos )
{
  int x, y, f;
  if(pos == postion_count ){
  find_count++;
  print_board();
  return;
 }
  x = postion[pos];
  y = postion[pos + 1];
  f = GET_BIT(x, y );
  if(f){
  int n;
  for (n = 1; n <= SIZE; n++){
      if(f & biton[n]){
         write_number(n, x, y);
         search(pos + 2);
         delete_number(n, x, y);
      }
    }
  }
}
```

関数search()を実行する前に確定サーチを行います。そのときに，確定できなかった場所は配列postionに，個数はpostion_countに格納します。あとはsearch()で，これをすべて確定すればいいわけです。

プログラムのポイントは再帰呼び出しです。数字nを置くことができる場合，write_number()で数字を書き込みます。そして，次の場所の数字を決めるため，search()を再帰呼び出しします。再帰呼び出しから戻ってきたら，delete_number()で数字を取り消して，次の数字を選びます。これで置くことができる数字をすべて試すことができます。

search()の引数posの値がpostion_countと同じ値になれば，すべての場所を確定することができたので，関数print_board()で盤面を出力します。ここで再帰呼び出しが打ち切られ，これ以上search()を呼び出すことはありません。

関数を再帰呼び出しする場合，このような停止条件が必要になります。もしも，停止条件を忘れたり，設定した条件を満たさない場合は，再帰呼び出しが止まらず，C言語では暴走することになります。ご注意くださいませ。

このように，再帰呼び出しを利用すると，バックトラックは簡単に実現することができます。ただし，再帰呼び出しに慣れていないと，このプログラムを理解するのは難しいかもしれません。納得できない方は，人間トレーサーになってプログラムの動作を追いかけてみてください。

ところで，松田晋氏が作成されたプログラム[4]では，バックトラックの中で確定サーチを行っています。これに対し，このプログラムは確定サーチで数字を決定し，それでも決まらない場所に対して，単純にバックトラックで数字を決定します。確定サーチだけでも解ける問題がありますし，たとえ数字が決まらなくても，確定サーチを行うことで可能性のある数字を減らすことができるので実行時間は速くなります。

＊　＊　＊

どのプログラミング言語でもそうですが，上達の秘訣は実際にプログラムを作ってみることです。ところが，いざとなると「さて，なにを作ろうか？」と困ってしまう方も多いのではないでしょうか。このようなときにぴったりの題材が「パズルの解法」です。なんといっても，実際にパズルが解けたときの喜びは大きく，プログラムを作る意欲をかきたててくれます。このプログラムに興味を持った方は，ほかのパズルにも挑戦してください。

参考文献

[1] 佐々木崇「ナンバープレイス」月刊・電脳倶楽部 Vol.99 満開製作所
[2] 中村隆生「ナンバープレイス解法プログラムを作れ」月刊・電脳倶楽部 Vol.102 満開製作所
[3] 舩本昇竜「ナンバープレイス自動解法プログラム」月刊・電脳倶楽部 Vol.102 満開製作所
[4] 松田晋「実践アルゴリズム戦略 解法のテクニック<第11回>バックトラックによる数独の解法」C MAGAZINE 1993年3月号 ソフトバンク

Direct3D Retained Mode講座
お魚を泳がせる

菊地 功　Kikuchi Isawo

今回から応用編だ。Retained Modeの範囲内で実際にいろいろな処理を行ってみたい。最初に2D画像から3Dデータを生成して，魚らしく泳いでいるようなモーションをつけてみよう。地道に頂点変換を行っても意外とすんなり動いてくれることがわかるだろう。

この連載も3回目である。だいたいいままででRMの基本は押さえられたと思われるので，今回はちょっと趣向を変えて，RMを応用したアプリケーション作成例を挙げてみよう。お題目は「2Dから3Dへ」。題材としては，前号のVisual Laboratoryから川原氏の「シーラカンス」を取り上げる。というか，このシーラカンスを見てこのネタを思いついたんだけど。要するに，2D画像をパクッと食べさせると3Dになって動き出したらいいなってハナシである（昔そういうソフトもあったような気もするが）。

厚みをつける

魚ってのは，側面から見ればほぼすべての形状が出てるし，断面を見ても，形状の入り組みはほとんどない。つまり，形状的にいうと側面からの型抜きに適している。タイヤキみたいなもんだ。だから，側面から見た輪郭があれば，あとは厚みさえどうにかすれば，それらしい3D形状が作成できる。方法としては，たとえば断面が楕円となるように頂点を補間してやるという手があるが，それではヒレなども膨らんでしまい，うまくない。

そこで，今回は2Dでの作業がひとつ増えるが，ハイトマップを使うことにした。ハイトマップとは，高い部分ほど明度の高い色でペイントした画像である（一般的にはグレースケール）。側面から見たハイトマップを作ってそれを厚みとし，ついでに厚み0の部分とそれ以外の部分の境界を輪郭としてしまえば一石二鳥だ。

方法としては次のようになる。まず魚の上にメッシュ状の格子を敷き，その格子の交点の高さをハイトマップ（図1）から取得，四角形ポリゴンを並べていく（図2）。ただし，このままでは半分壁に埋め込まれた化石になってしまうので，輪郭に沿って縁を切り取らなければならない。ここで，ある格子が輪郭によって一部を切り取られている状況を考えよう（図3）。まず4頂点の高さをハイトマップから取得する。このとき，頂点1は高さ0，すなわちオブジェクトの外であるから，この頂点にはポリゴンは貼られない。その隣接頂点はというと，頂点2は高さがあるので，頂点1と頂点2の間には輪郭との交点があると判断し，その交点を探して新しい頂点とする。

反対に，頂点4は高さ0なので，ここも頂点からは除外し，さらにその隣接頂点3を見ると，ここは高さがあるので同様に頂点3・4間に新しい頂点を設ける。こうして新しくできた頂点を結んでポリゴンとすればいい（輪郭との交点にできた頂点は高さ0とする）。この方法だと，図4のような状況ではあまり正確に輪郭をトレースできないが，そのような場合にはメッシュを細かくして対処することになる。ちなみに，この方法ではポリゴンは図5のような状況で最大6頂点である。

この辺りのプログラミングは力技になるが，隣接格子のポリゴンと頂点は共用したいので，1ライン分のバッファを用意して以前の頂点番号を保存し，それを利用することで同じ頂点を複数登録することを避けている。ここでは詳しくは述べないので，ソースを見てよく考えてほしい。また，実際のポリゴンの生成は，IDirect3DRMMeshBuilder3のAddFaces()で行っている。

```
HRESULT AddFaces (
  DWORD dwVertexCount,        // 頂点数
```

図1　ハイトマップ。見やすいように強調してある。黒い部分にはポリゴンは張られない

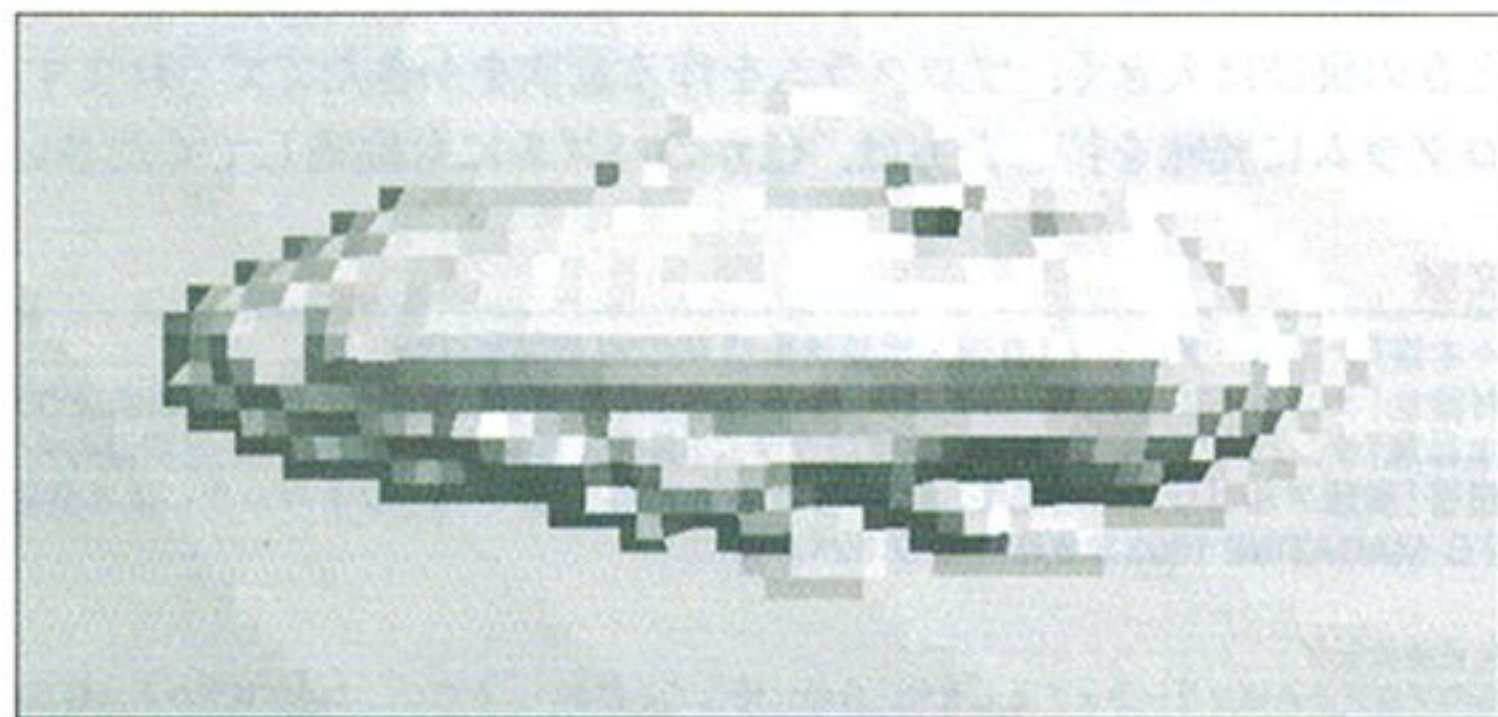



図2　ハイトマップを参照してポリゴンを並べた

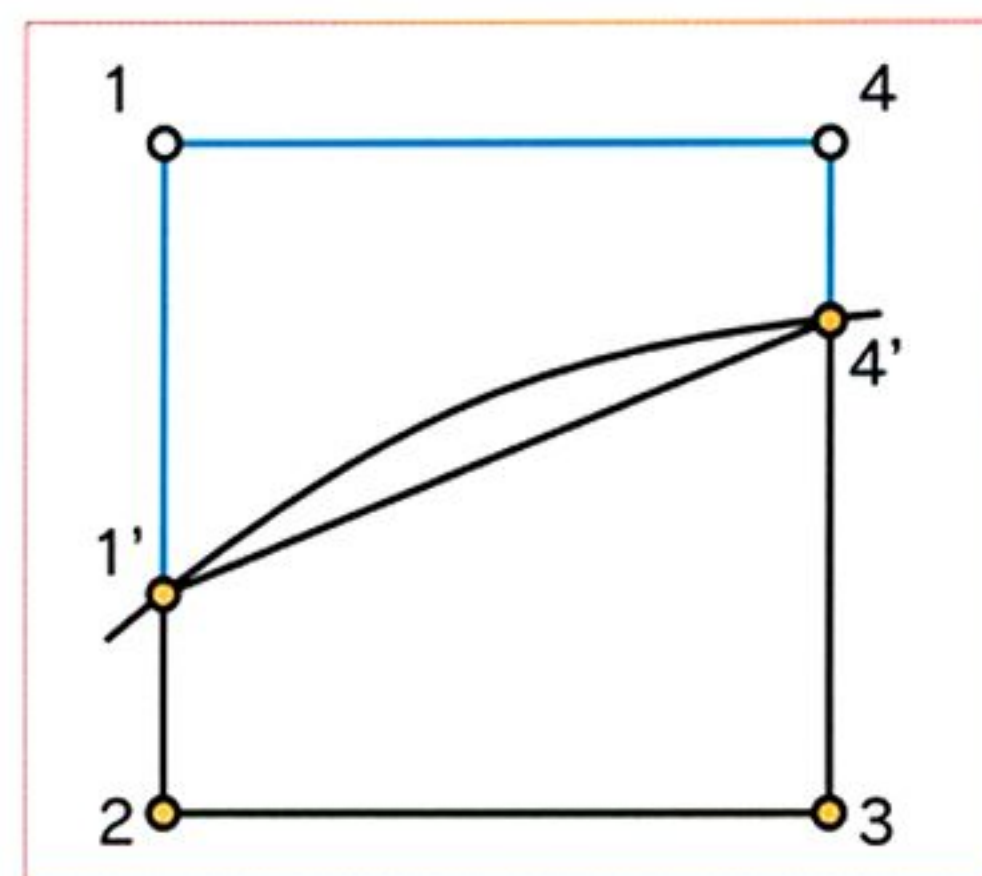



図3
輪郭によって切り取られたポリゴン

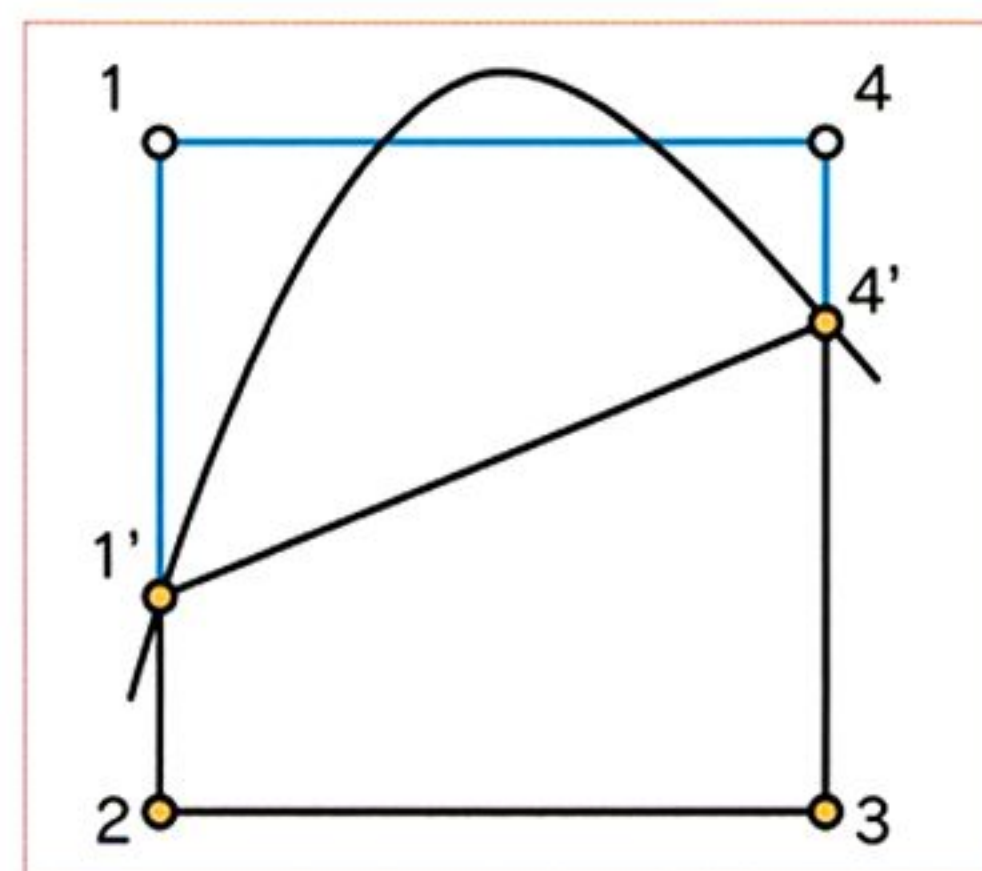



図4
このような輪郭は，頂点1と2の間に交点があると認識されない

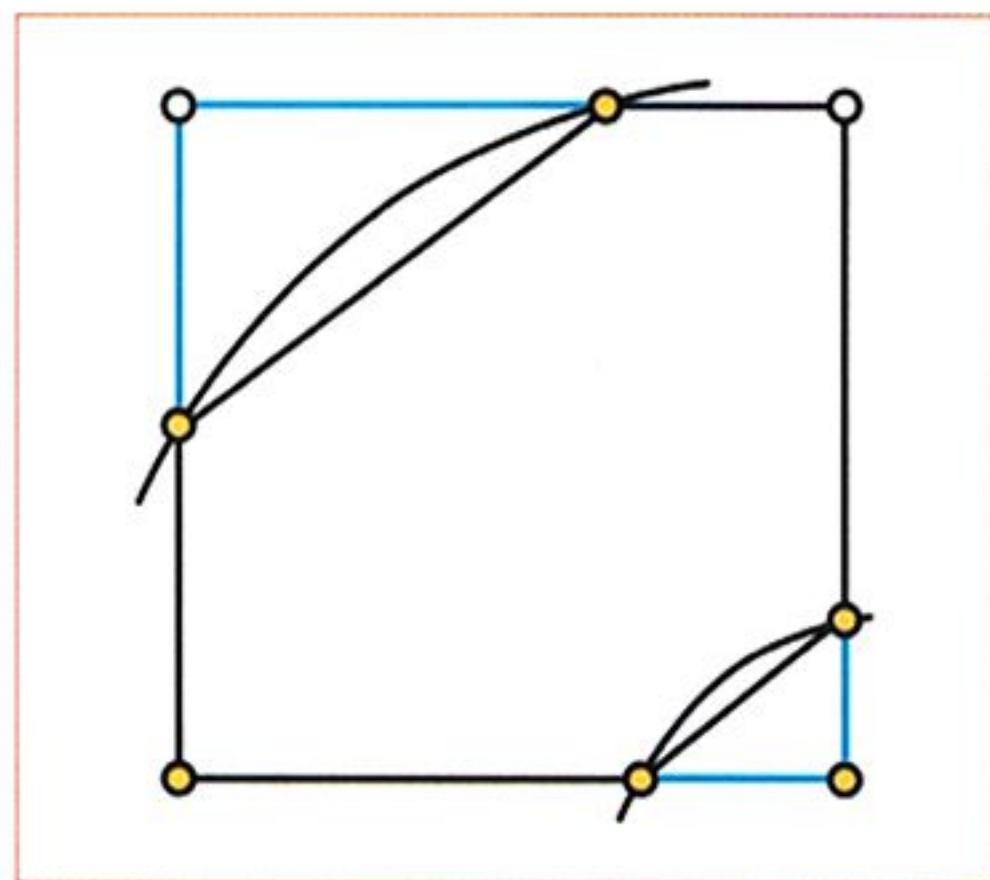



図5
ひとつのポリゴンの頂点数は最大で6

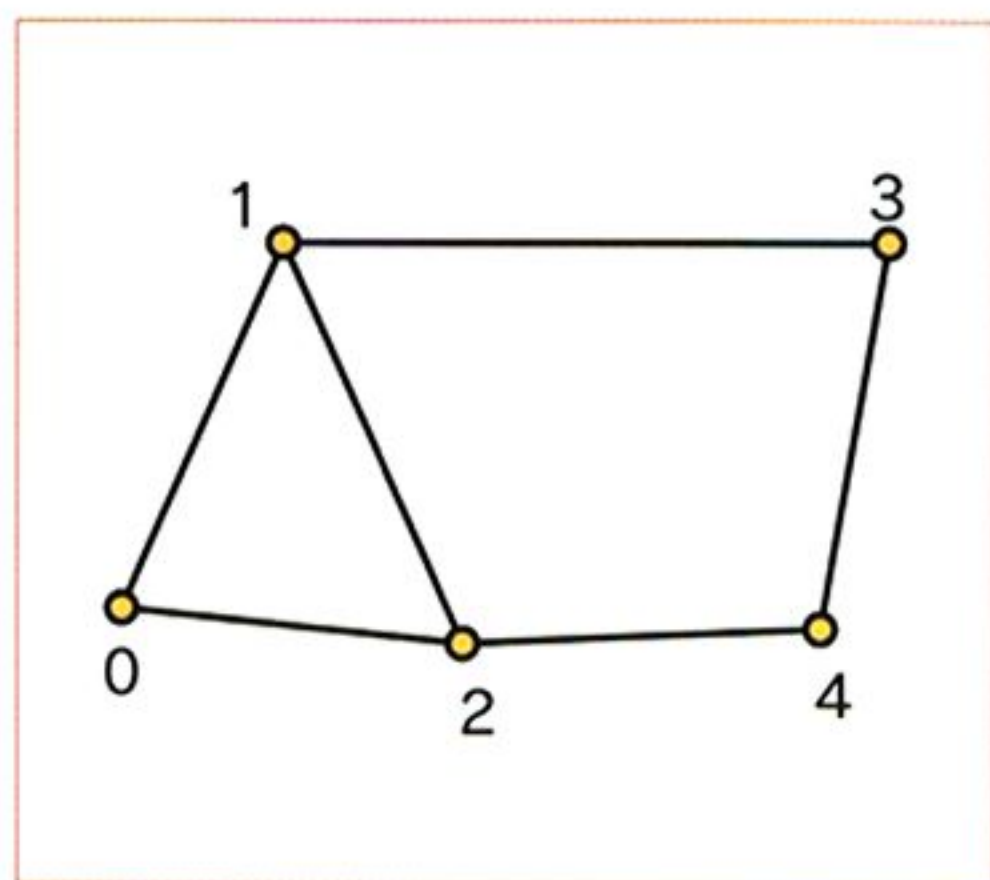



図6
面データの記述

```
    D3DVECTOR * lpD3DVertices,  // 頂点を格納した配列のアドレス
    DWORD normalCount,          // 法線数
    D3DVECTOR * lpNormals,  // 法線を格納した配列のアドレス
    DWORD * lpFaceData,         // 面データを格納した配列のアドレス
    LPDirect3DRMFaceArray * lplpD3DRMFaceArray
                        // 作成された面のフェースアレイが返るポインタ
  );
```

とりあえず法線はここでは作らないので，normalCountは0，lpNormalsはNULLとする。また，フェースアレイも必要ないので，lplpD3DRMFaceArrayもNULLだ。問題となるのは，lpD3DVerticesとlpFaceDataの内部構造だろう。lpD3DVerticesの方は，単なるD3DVECTORの配列であるので，そのメンバであるx，y，zに頂点座標を放り込めばよい。このとき，それぞれの頂点には0から順にインデックスが振られる。lpFaceDataには，頂点の数と頂点インデックスがワンセットで1枚のポリゴンを表す。また，ポリゴンリストは0で終了する。たとえば，図6のような2枚のポリゴンを作るならば，次のようになる。

```
    3,              // 頂点数3
    0, 1, 2,        // 頂点インデックス
    4,              // 頂点数4
    1, 3, 4, 2,     // 頂点インデックス
    0               // 終端
```

このようにして余計な部分を切り取ったのが図7である。ヒレの隙間に余計なポリゴンが張られてしまっているが，……あとで気がついた。これは図5の逆の現象である。つまり，ポリゴンが2つの三角形に分断されてしまわなければならないのだが，ひとつの六角形と認識してしまっているのだろう。ま，手遅れということにして，細かいことは気にしないことにする。

このままでは魚の半身だけなので，頂点をいじってもう一度AddFacesし，もう半身も作ってやる。ただし，その場合はポリゴンを形成する頂点の向きも逆回りになり，裏返ってしまうので，頂点インデックスの記述順も反転させる。こういったことを自動でやってくれるメソッドがあってもよさそうなものなのだが，私が探した限りでは見つからなかったので，これも手抜きだ。ついでにここで法線も立てておこう。IDirect3DRMMeshBuilder3には，自動的に法線を立ててくれるという，ありがたいメソッドが用意されている。

```
    HRESULT GenerateNormals (
      D3DVALUE dvCreaseAngle,
      DWORD dwFlags
    );
```

dvCreaseAngleは，ヘルプでは新しい法線を生成できる面同士の最小可能角度とある。つまり，面同士のなす角がこれより小さければ，法線を立てない，つまり折り目を出す，ということだと思うのだが，どうもうまく制御できない。なにかが間違っているような気がする。結果を見ながら適当な値を放り込むことにする。dwFlagsは，次の2つのフラグを指定する。

・D3DRMGENERATENORMALS_PRECOMPACT
法線を生成する前に，同じ座標の頂点があればひとつにまとめてくれる。面の生成時に頂点がダブらないようにしたが，左右の半身をくっつけたときに，輪郭上の頂点は左右で同じであるので，その頂点をまとめることができる。

・D3DRMGENERATENORMALS_USECREASEANGLE
dvCreaseAngleを有効にする。

ヘルプには，このうちどちらか一方を指定するとある。dvCreaseAngle

図7　図2から壁の部分を切り取った

図8
GenerateNormals () でD3DRMGENERATENORMALS_PRECOMPACTだけを指定した場合。左右の境界の頂点がマージされていない(法線が別々)なため，明度の境界がはっきりと現れる

の制御法もよくわからないので，D3DRMGENERATENORMALS_PRECOMPACTにしてみたのだが，なぜかちゃんと頂点をマージしてくれない（図8）。しようがないので両方指定してみたらうまくいった。謎だ。

テクスチャを貼る

さて，立体化はできたので，次はテクスチャだ。側面から平面ラップすればいいのだが，その場合だと位置やスケール合わせが面倒である。ハイトマップとテクスチャをぴったり重ねて作っていれば，むしろ手計算のほうが楽だ。ここではあとあとのことを考えて，まずIDirect3DRMMeshBuilder3からCreateMesh（）でDirect3DRMMeshオブジェクトを作ってから，テクスチャ座標を設定することにしよう。

```
HRESULE CreateMesh
        ( LPDIRECRT3DRMMESH * lplpD3DRMMesh ) ;
```

これだけで，Direct3DRMMeshBuilder3オブジェクトが現在抱えているポリゴンを，そっくりDirect3DRMMeshに移すことができる。そしたら，その頂点情報を取得する。

```
HRESULT GetVertices (
  D3DRMGROUPINDEX id,
  DWORD index,
  DWORD count,
  D3DRMVERTEX * returnPtr
 ) ;
```

idとは，取得する頂点のグループIDであり，Direct3DRMMeshBuilder3からCreateMesh（）で作成された頂点はIDが0となる。indexは取得する最初の頂点インデックス，countは取得する頂点の数であるから，indexは0，countはそのグループに含まれる頂点数を指定すればよい。頂点数はGetGroup（）メソッドで取得できる。returnPtrは，データを格納するD3DRMVERTEX構造体へのポインタだ。

図9
D3DRMGENERATENORMALS_PRECOMPACTとD3DRMGENERATENORMALS_USECREASEANGLE双方を指定すると，なぜかうまくいく

```
typedef struct _D3DRMVERTEX{
    D3DVECTOR position;
    D3DVECTOR normal;
    D3DVALUE  tu,  tv;
    D3DCOLOR  color;
} D3DRMVERTEX;
```

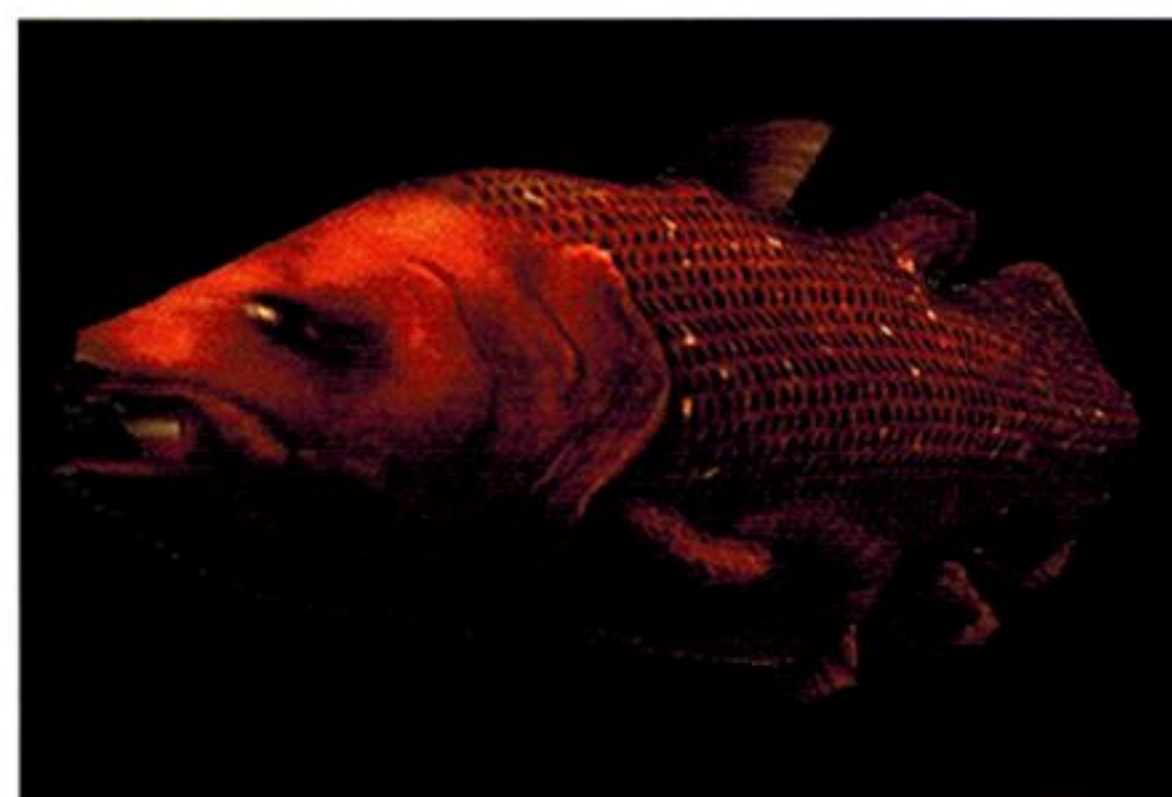
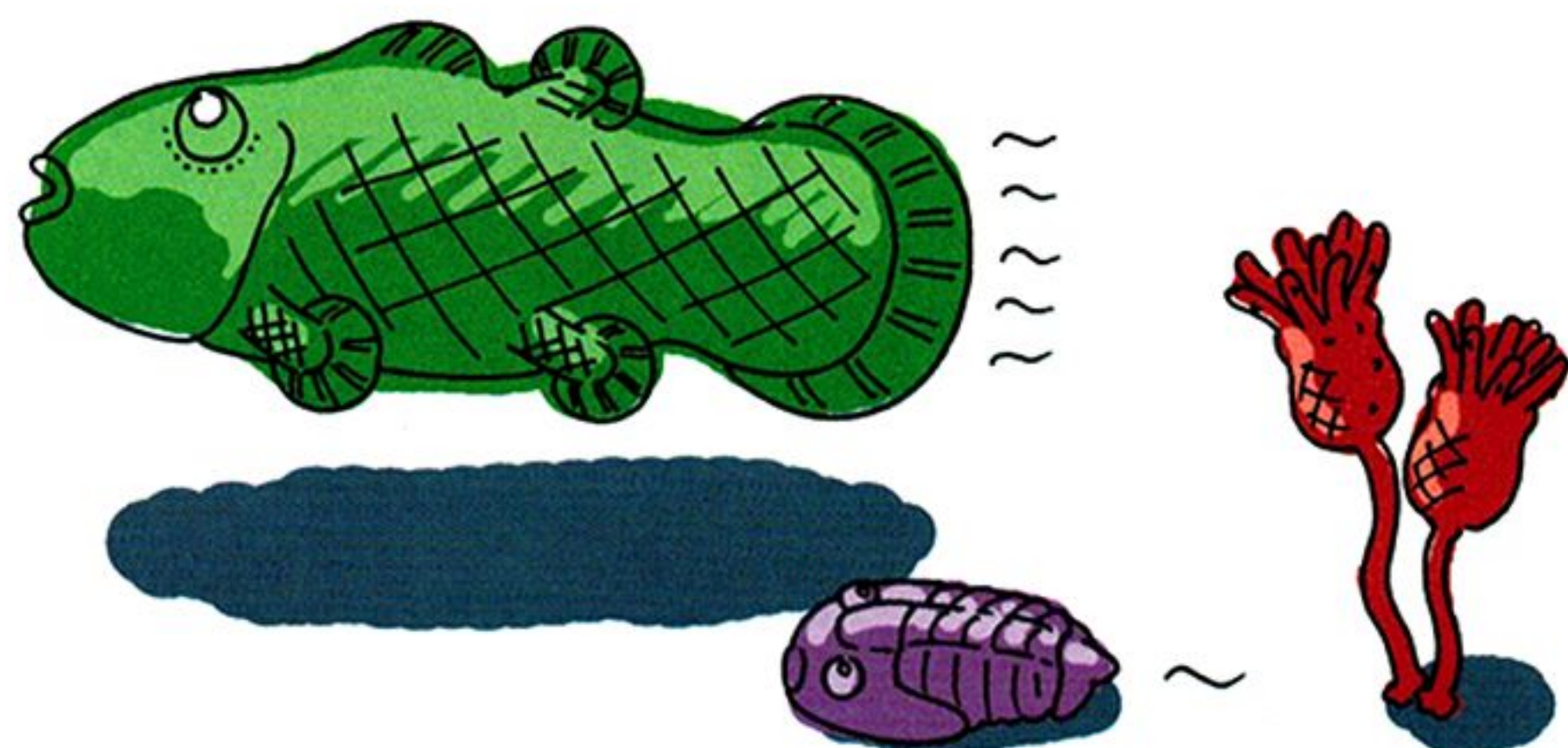
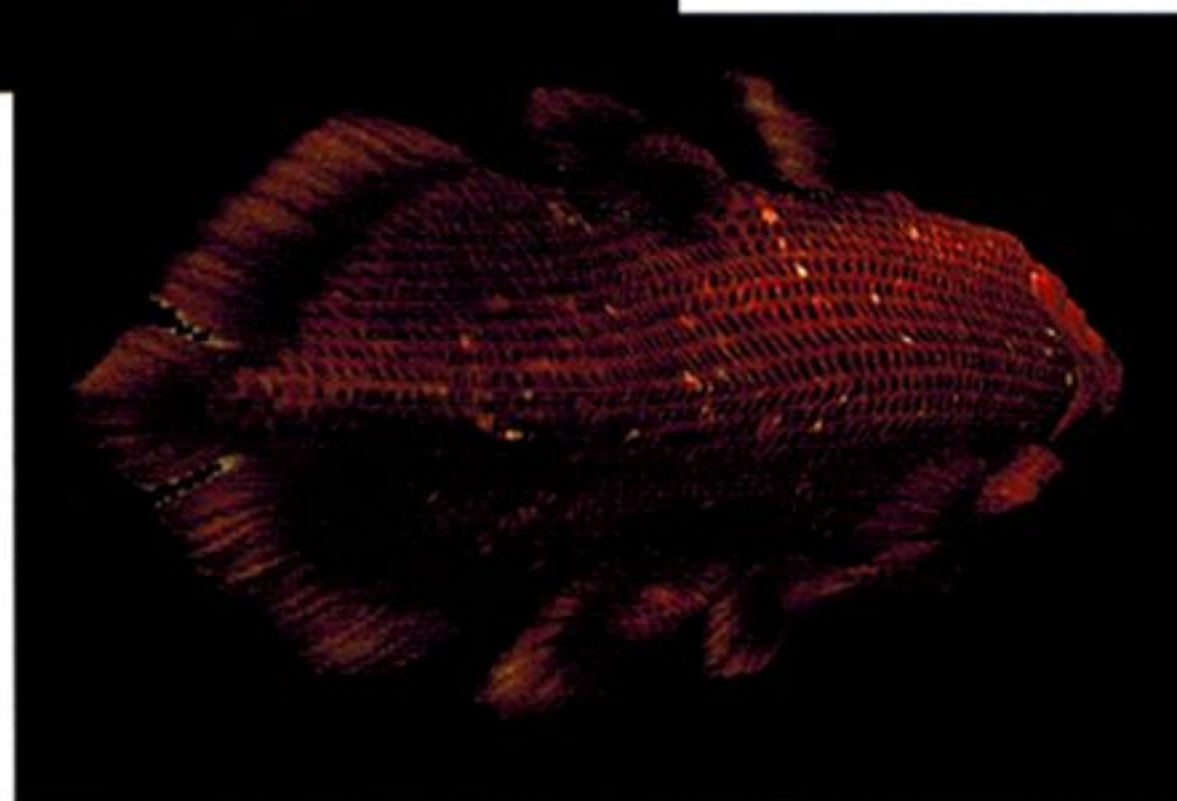



図10，11
立体化されたシーラカンス

positionは頂点の座標，normalは法線，tu，tvはテクスチャ座標，colorは頂点の色である。つまり，positionからtu，tvを求めてやればよい。tu，tvは0から1までの値であるから，ある頂点に画像内でx，yの座標の点をマッピングしたければ，

```
tu=x/width;
tv=y/height;
```

となる（width，heightは画像の縦横ドットサイズ）。こうしてテクスチャ座標を設定したら，今度はSetVertices（）メソッドでメッシュに頂点データを設定する。これで立体シーラカンスの完成だ（図10，11）。

泳がせる

とはいえ，このままでは剥製でしかない。くるくる回して観賞するのもいいが，ここは一発「生きた化石の復活」ってことで，思いきって泳がせてみたい。基本的に魚のモーションなんて，背骨をサインカーブに沿ってくねらせてやればいいだけなんじゃないかと思う。ただし，だからといって頂点を横にスライドさせるだけでは不自然で，それに加えて体の軸を中心にして，前後にも回転させる必要がある（図12）。さらに，法線にも同様の回転を与える必要がある。それを頂点すべてに対して行わなければならない。なかなか面倒なことだ。そうすると頭蓋も変形してしまうのだが，それは気にしないことにする。頂点と法線の計算方法は，ごちゃごちゃしてしまうので，ここでは詳しくは述べない。自分で図を描いて考えるなり，ソースを参考にするなりして考えてほしい。問題は，どうやってそのモーションを実現するかだ。前回試したインターポレータは，頂点の数と制御のポイントが多すぎて，あまり向いているとはいえなそうだ。こういう場合は頂点バッファが適切であると思われるが，RMではサポートされていない。そこで，また力技である。先ほどのSetVertices（）メソッドを使って，リアルタイムで頂点を制御することにしよう。

演算は，マシンやビデオカードの性能によって泳ぐ速度がまちまちにならないように，時間管理をしている。しかも，ほとんど手抜きなしに，1点ずつ大まじめに三角関数を使って小数点演算しているために，めちゃめちゃ遅くなるだろう，と思っていたのだが……最近のCPUのパワーはやはり恐ろしい。確かに私の環境（Celeron300A/450MHz＋G200）は一般的な平均よりは上だと思うが，ストレスなしにスムースに泳ぎ出してしまった。確かに頂点数が極端に多いわけじゃないけど……動くか？　普通。フルスクリーンモードなら，平気で60fps近く出ているように見える。こういうの見ちゃうと，なんかこれ以上チューニングする気も失せちゃうよな。

ってことで，もうこれでいいや（図13，14，15）。テクスチャもうねうねと伸縮して，結構気持ち悪い。画面写真ではこのうねうね度はわからないだろうから，各自で実行して試してもらいたい（図16）。ハイトマップやテ

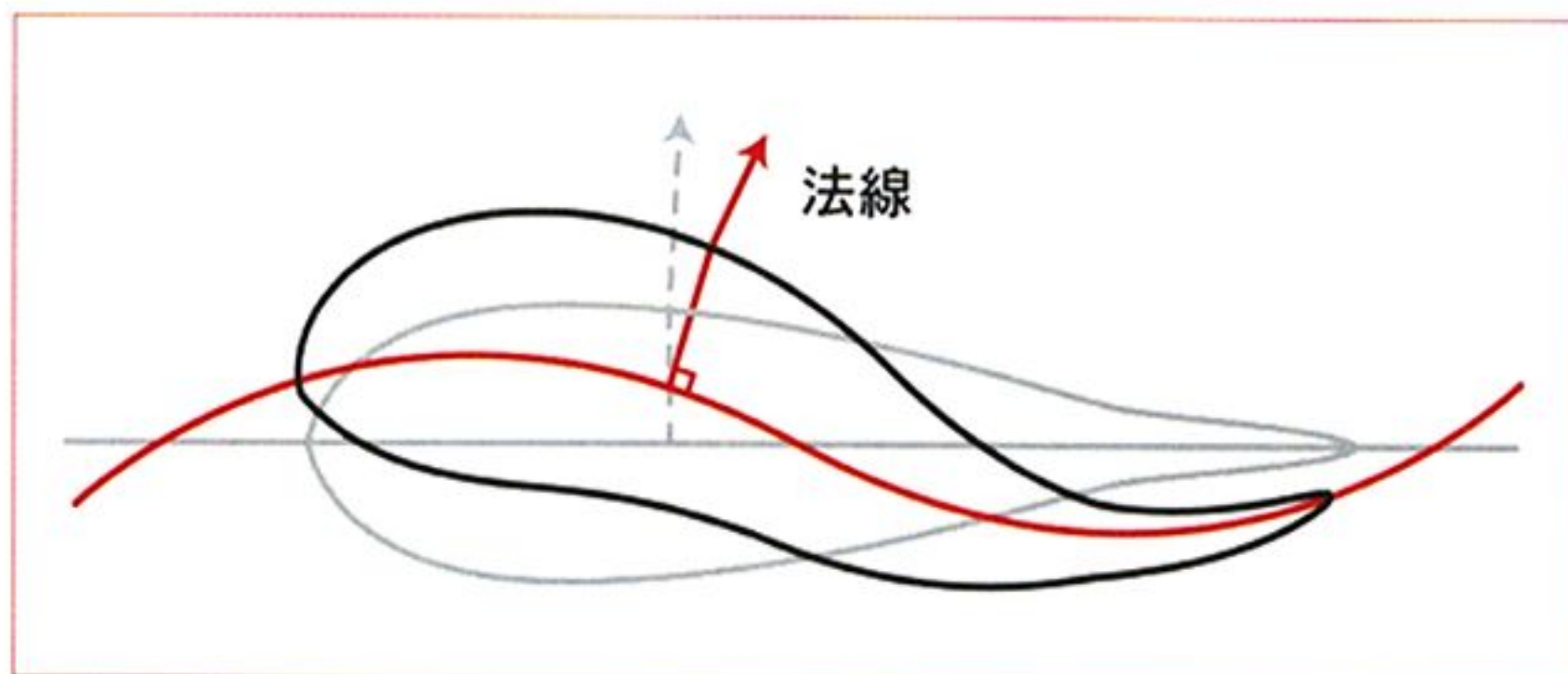



図12 背骨のくねりによる表面頂点の移動

図13, 14, 15
泳ぎ出したシーラカンス。本物はこんなに活発じゃないと思うけど

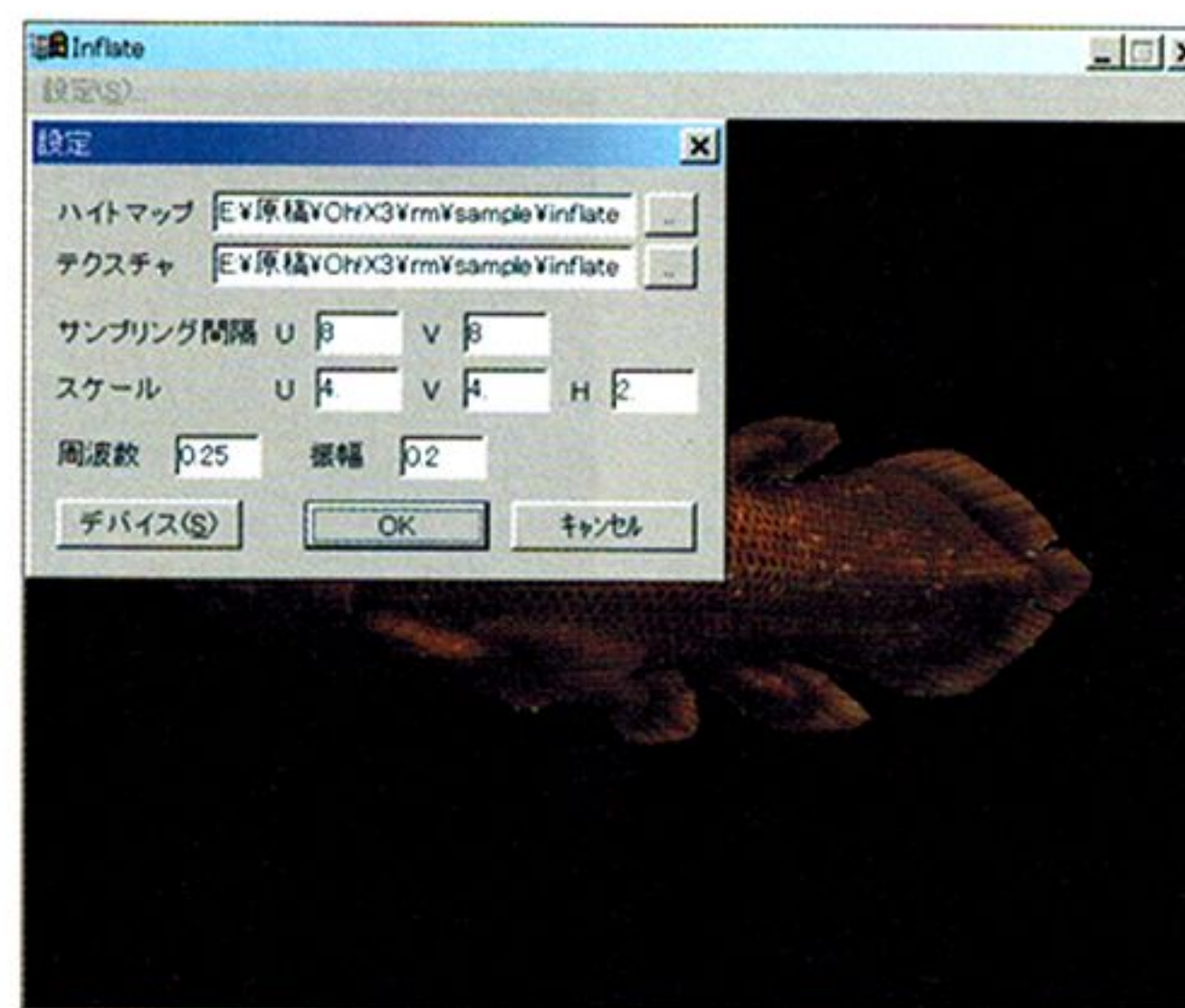



図16
サンプルInflate。ハイトマップやテクスチャをはじめ，パラメータを設定できる

図17
ウナギのようなシーラカンス

図18 サンゴヘビモドキ

クスチャは各自で用意したものを使用できるように汎用的に作ってある。ただし，ハイトマップは8ビット非圧縮で作ること。グレースケールでなくても構わないが，パレットは見ておらず，パレットのインデックス値の大きいものほど高いとみなしている。また，テクスチャの制限により，両方とも一辺が2の累乗，あるいは640×480ドットのサイズで画像を作ること。もちろんハイトマップやテクスチャは同じサイズにすべし。ほかにもいくつかのパラメータを指定できるようになっている。

・サンプリング間隔
　ポリゴンの頂点の間隔。Uは幅，Vは高さをドットで示す
・スケール
　拡大率。Uは横方向，Vは縦方向，Hは厚さ
・周波数
　大きいほどくねりが激しい
・振幅
　背骨を左右に振る幅

　サンプリング間隔以外，数値自体にはいろいろな補正値が含まれていて，あまり意味はない。ある基準に対する相対値だと思ってもらいたい。たとえば，周波数と振幅を大きくすると，図17のようにすごいことになる。蛇なんかもできそうだ(図18)。また，カーソルキーでオブジェクトの上下左右回転，シフトキーを押しながらだと左右や前後移動ができるようになっている。

　このアプリケーションは，特にXファイルを吐いたり，モーションを保存したりできるわけではなく，ただ眺めて楽しむだけだ。しかし，これをもうちょっと改良してスクリーンセーバーを作るとか，ゲーム(にはちょっと重いか)にするとか，そういったものの基礎にはなるんじゃないかと思う。力技だけで押し通しても，なんとかなることもわかったし。

筆者急病により……

　ごめんなさい。今号はIMはお休み。決して飽きたとか挫折したとかじゃなくて，時間がね，足りないのよ，圧倒的に。その代わりといってはなんだけど，重大発表をしたい。Direct3Dといえば，やっぱ最終的にはゲームだろってことで，IMで本格的なゲームを作る予定。それも，本誌の付録とか，Mookとかじゃなくて，無謀にもちゃんとしたパッケージソフトにしたいと。本誌では，次号からそのメイキングみたいなことを数回にわたってやる予定。ジャンルは，ポリゴン恋愛シミュレーション，じゃなくて，バリバリのシューティングゲーム。とりあえず案は松竹梅くらい出てるんだけど，どれになるかは私の能力と時間で決まってくるでしょう。なんとか年末商戦には間にあわせたいぞ，と。来年のね。ということで，キャラクターグラフィッカー募集。え？

project team DoGA

CGアニメにおける ストーリーのひねり出し方

かまたゆたか

第１章：ストーリーの基礎

●才能がなくてもストーリーはできる

当チームで主催しているCGAコンテストの応募数が倍増したり，アマチュアのCGアニメにスポットをあてたテレビ番組がいくつか放映されるなど，自主制作CGアニメもやっと市民権を得てきたように感じます。以前はCGアニメというと，かなりマニアックな趣味というイメージがありましたが，もはやもの珍しいという感じはなくなってきました。

しかしながら，CGアニメに興味はあっても，まだ制作には踏み切れないという方が多いのも事実です。そのような方を対象にアンケートを実施すると，原因は大きく分けて３つあることがわかります。

1）制作している時間がない

こればっかりは，当方ではいかんともしがたいので，各自で工夫してください。どんなに多忙でも，制作している人は，文字どおり寝食を削って制作しているということを忘れずに。

2）CGソフトが高価，難しい

この点については，最近大きく改善されてきています。ソフトはどんどん安くなってますし，初心者向きのものも出てきました。また，解説書やCG専門誌の増加も一役買っています。当チームでも，本誌復刊１号で紹介した初心者向けCGアニメ学習ソフト「DOGA-L1」をフリーウェアとして配布するなど，入門環境の改善に努めています。

3）ストーリーが思いつかない

上記の２つに対して，これは意外な答えでした。私などは，作りたくてもその機会がないまま

眠っている脚本が40本ぐらい溜まっています。“ストーリーぐらい，なんぼでも思いつくやん”と思うのですが，“じゃぁ，その方法を説明してくれ”といわれると，これはなかなか難しい問題です。ついつい“なんとなく”とか，“急にひらめいた”とか，なんの参考にもならない答えをしてしまいます。

ではやはり，ストーリーを作るには，ひらめきや才能といった天性的なものが必要かといえば，そうは思いません。というのは，以前“映像”について同様の疑問を感じ，いろいろ検討した結果，一見ひらめきによって制作されているかに見える映像も，大部分が単純な法則に則って構成されていることがわかりました(ソフトバンク発行「CGアニメ完全入門ガイド」参照)。ただ，優れた方々はその作業を頭の中で無意識に行うので，傍目から見ると特殊な才能に見えてしまうだけなのです。

同様にストーリーの場合でも，“なんぼでも思いつく”という人は，無意識下にストーリーを創作する方法を確立しているにすぎないと思うのです。その思考を，ほかの方々にもわかるように文章にしてまとめてみようというのが今回の試みです。脚本に関する専門書を読み，ほかのCGアニメ作家の方々の意見を伺い，そして自分の経験をベースにまとめてみました。ストーリーの段階で行き詰まっている初心者の方の参考にでもなれば幸いです。

●CGアニメの制約

ストーリーを創作する方法自体には，CGアニメ，実写映画，演劇などでも大差ないのでしょうが，現実問題としてCGアニメにはいくつかの制約があります。ストーリー一般を語る前にCGアニメならではの注意点を解説します。

まず重要なのが作品の長さです。個人レベルで制作することを考えると，そんなに長いストーリーは実現できません。CGAコンテストの入選作を見ても，最長15分，平均は5分ぐらいです。初心者でしたら，まずは2分ぐらいの作品を制作すべきです。“いや，どうせ作るならすごい作品を……”などという邪なことを考えると，さんざん苦労した挙げ句，途中で挫折することになるのです(このパターンで，どれほど多くの初心者がCGアニメ界から消えていったことか……)。

さらに述べると，頭の中の構想を実際に制作すると，たいてい2倍ぐらいの長さになってしまうものです。ですから初心者は，30秒ぐらいのストーリーを考えれば，実際に制作すると1分になり，説明不足でつながらないカットを補充し，タイトルやエンディングクレジットなどを入れるとだいたい2分の作品ができあがります。

“いくらなんでも30秒とか2分とかではストーリーにならない”と思う方も多いでしょうが，それは劇場映画やテレビアニメに影響されています。たとえばテレビのCMは，商品名などを含めてたったの15秒ですが，ストーリー性のあるものはいくらでもあります。その数倍の時間があるのですから，かなりのものが描けるはずです。まずは，劇場映画とテレビアニメを頭の中から追い出し，ちょっと長めのCMを作るんだと考えてください。

次にモデリングの問題です。たとえば，3世紀の長安が舞台となるようなストーリーを考えても，長安の街をまともにモデリングしたらそれだけで数年かかります。同様にノアの箱船をモチーフにしたストーリーを考えても，100種類の動物をモデリングするのはやっぱり無理です。CGA制作においてモデリングの作業量はかなりの比重を占めるので，ストーリーの段階から，モデリングが簡単かつ少なくて済むように考慮しなければいけません。

いちばん楽な舞台は，宇宙空間といえます。背景を作る必要がありません。もっとも，これだけでは絵的に寂しくなってしまいますが……。ですから，基地の中とか，ある部屋の中とか，閉鎖された空間の中で完結するストーリーがおすすめです。また，木もろくに生えていない荒野を舞台にするという手もあります。逆にたくさんのシーンがあって，舞台がころころ変わるのは，極力避けたほうがよいでしょう。

●基本構成

ストーリーは実に多種多様で，例外も多く存在するのですが，構成の基本といえば“起承転結”です。

○「起」＝「状況」と「発端」

「状況」とは，ストーリーを理解するために事前に知っておく必要な情報です。舞台はいつなのか，どこなのか，どういう世界なのか，主人公は誰で，どんな人間なのかなどを視聴者に提示します。その次に，ある事件をきっかけに，主人公が成し遂げなければならない目的を持ちます。これが「発端」であり，主人公が行動を起こすとともに，ストーリーが動き出します。

○「承」＝「障害」と「葛藤」

主人公がなんの障害もなく目的を達成してしまうと，ストーリーは成立しません。主人公の行動に対してさまざまな障害が現れ，そして葛藤しながらそれを乗り越えていく主人公を描きます。劇場映画のように長い作品の場合，この障害が大きく，数も多くなってきます。

○「転」＝「クライマックス」

主人公は，「起」で提示された最初の目的を達成するため，最大の障害に立ち向かいます。そして，見事問題を解決します。

○「結」＝「テーマの定着」

主人公の達成を通じて，このストーリーのテーマが視聴者に賛同を得て，話は終わります。「結」は，「起」「承」「転」と比較するとかなり短いのが普通です。

多くのストーリーがこのような構成になっていることは，例を挙げるまでもないでしょう。起承転結を明確にすることで，わかりやすく，見ていて納得のいくストーリーを作ることができます。

第2章:ストーリー作成の流れ

●流れの概要

ストーリーを練る手順を順番に解説していきます。まずは，その流れをひととおり把握してください。

1)ネタを集める
　ネタ，つまりストーリーの基になる断片的なアイデアを集めます。そのなかから関連のありそうなものを探します。
2)本筋を作る
　ストーリーをたったひとつの文で簡単に書いてみます。
3)テーマを決める
　この作品を通じて，自分は視聴者になにを訴えたいのかを考えます。
4)あらすじを作る
　半ページぐらいにストーリーの概略をまとめます。台詞などはまだ必要ありません。
5)設定を用意する
　登場人物や舞台などについて，細かい設定を用意します。
6)エピソードを考える
　主人公に与えられる障害とその解決策をいくつか考えます。
7)シナリオを書く
　すべての台詞まで決め，シナリオという形で文章にします。
8)推敲する
　より面白いストーリーにするために，推敲を重ねます。

以下，各ステップを詳しく解説します。

●1.ネタを集める

具体的なネタの集め方については，別枠にまとめましたので，そちらをご覧ください。ネタが見つからないと嘆いている人の大部分は，ネタを大げさに考えすぎています。別に人類存亡の危機や巨大企業の陰謀に巻き込まれた人しか作品が作れないわけではないのです。そんな発想は，ハリウッド娯楽超大作に毒されている証拠といえます。

ただ，出来事がそのままネタになるわけではありません。「抽出」や「誇張」をしてみましょう。つまり，その出来事のどこが重要かを考え，関連の薄い部分は削除し，強調される部分に都合のよい設定や結果をでっち上げるのです。

たとえば，
〝祖母の家の引っ越しの手伝いに行かされた。部屋の掃除やゴミのたき火を手伝わされて閉口し

ネタの探し方

column1

a)日常生活から

〝自分の日常はあまりに平凡で，映像作品のネタなどない〞と考えてはいけません。平凡な日常生活といっても，ムカッとしたり，ホッとしたり，少しぐらい感情が動くような出来事はあります。たとえば，11thCGA コンテストでいちばん人気だった「洗濯危機一発」は，〝コインランドリーにひとりでいたとき，1匹の蚊がうっとうしかった〞というだけのネタです。このように，どんなに些細な出来事でも，〝これはネタになるのではないか〞という目で見ることが大切です。

b)過去の体験から

平凡な人生を歩んできたという人でも，長い人生，めったにできないような体験のひとつや2つあるでしょう。これは格好のネタになりますので，よく思い出してください。自分の体験は，事実なので描写にリアリティが出ますし，自分だけしか知らないのでオリジナリティがあるのです。

ただ，体験ネタは，その件に関して詳しく知っているだけに，こと細かに説明してしまいがちです。その事件の核になる部分だけを取り出す抽出の作業が重要になります。

c)専門知識から

普通の人の知らないような専門知識があれば，ネタとして有望です。大学で専攻した分野や，趣味の知識などが役立ちます。すぐにストーリーに直結しなくても，たとえば犯人が使うトリックや，クライマックスの事件の解決法などに利用すれば，視聴者をあっといわせる意外性が生まれます。

d)別の視点で見る

物事をそのまま受け止めるのではなく，別の視点で見たり，想像することが大切です。たとえば，テレビで報道されている凶悪事件も，実はこの男は真犯人じゃないと勝手に想像すれば，ストーリーのネタになります。また，ロボットアニメも，描かれている主人公ではなく，敵の兵士や戦場の市民の視点で考えれば，まったく違うストーリーが生まれてくるでしょう。

e)夢から

ストーリーのことを常に考えていると，寝ているときに見る夢にもストーリー性が強くなるような気がします。夢は矛盾している点が多いという欠点があるものの，ときに奇想天外なアイデアが出ることもあります。ただ，夢はすぐ忘れてしまいがちなので，朝起きたらすぐにメモするように心がけましょう。

f)ほかの作品から

既存の映画やアニメ，漫画や小説も当然ネタになります。本文で述べたように，多くの視聴者が知っているような作品からネタを取ってくれば〝オリジナリティがない〞と罵られるのですが，逆にいえば，誰も知らないような作品ならバレないものです。〝バレなくても，オリジナリティがないのは同じだ〞と思うかもしれませんが，人が知らないような作品を知っているということ自体がその人のオリジナリティだとはいえないでしょうか。制作者には一般の人より広い見識が求められるのです。

g)ブレインストーミング方式

これは，ストーリーに限らずアイデアを出す一般的な方法です。数人が集まって，無責任に，いい加減なアイデアを，とにかく多数発言していきます。人の発言を受けて，追加，アレンジしてもいいし，まったく関係ない発言でも結構です。ただ，決して否定的な意見はいってはいけません。1時間も続けていると，不思議なことに，それなりのネタにまとまります。仲間で制作する場合には，一度試してみてください。

h)図書館の利用

図書館は，ネタを探すのになかなか効率のよい場所です。1日こもればかなり集まるでしょう。とにかく，なにか興味がある本があれば，目次で概要をつかみながら片っ端から斜め読みします。日頃，読む機会がないような本(例：歴史書，聖書，法律書，哲学書など）がおすすめです。逆に小説などストーリー性の高いものは，真剣に読み出すと時間がかかりすぎますので，敬遠すべきでしょう。

た。また，子供の頃はとても長く思えた廊下も，久しぶりに見ると普通の廊下だった”

というのが事実でも，「ノスタルジック」という点に注目すると，

・単なる引っ越しではなく，祖母が逝ったことにする(許せバァちゃん)
・20年ぶりに訪れたことにする
・部屋の掃除で閉口した部分は削除する
・ゴミの中から，幼少の自分が祖母に出した手紙が見つかることにする
・祖母との思い出のエピソードを考える

といったことをすれば，これで作品1本ぐらいできるネタになってきます。

それから，もうひとつ重要な注意ですが，人気のアニメやゲームからネタを取るのはやめましょう。話題になった映画やマンガも避けるべきです。なぜなら，そのネタについては，視聴者の多くが知っているので，せっかく作品が完成しても，パクリだの，オリジナリティがないだのと罵られることになるからです。また，ほかの人が似たような作品を作っている可能性も高く，視聴者側は，“またこれか”と思ってしまいます。

ネタがたくさん溜まってきたら，ネタの中でつながりがありそうなものを探してみましょう。たとえば，「悲しい」とか「楽しい」といった感情のつながりでもいいですし，「SF」といったジャンルで分類しても結構です。いくつかのネタのつながりが見えてきたら，さらにそれにつながるようなネタを意識して探していきます。こうして，ネタの固まりができてくるのです。

●2.本筋を作る

まず，本筋をいくつか作るところから始めましょう。本筋というのは，ストーリーをたったひとつの文で簡単に表現したものです。

たとえば，

・お人好しの刑事が悪徳企業を叩き潰す
・事故によりサイボーグとなった女が人間性を取り戻す

といった感じです。ここで重要なことは，「誰が」「なにをした」という2点を必ず盛り込むということです。ですから，

・トルコに伝わる小石を積み上げて作られた山の伝説

というのはいけません。この場合，

・古代トルコの小国の王が，小石を積み上げて巨大な山を築く

と書きます。

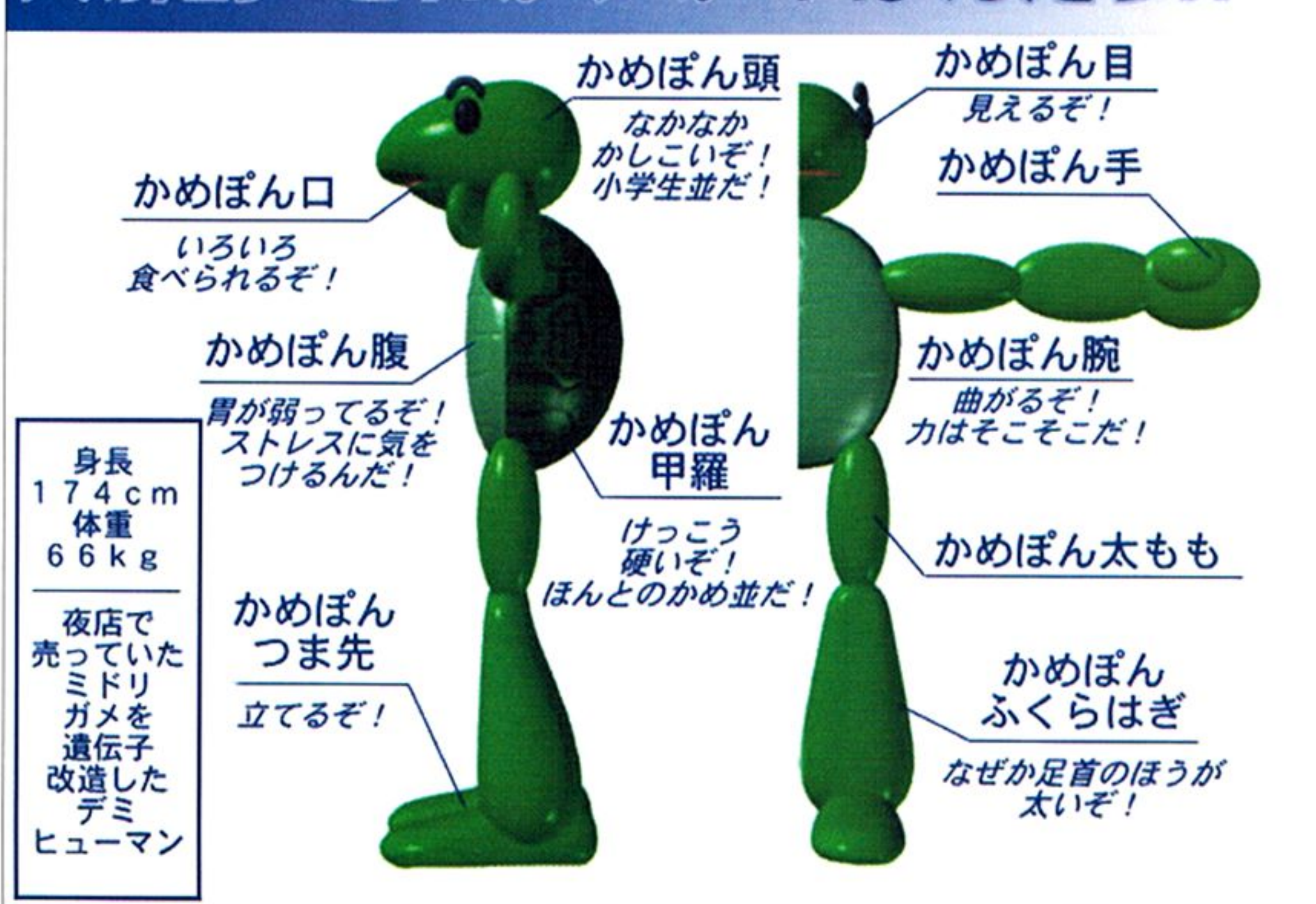
本筋は，あまり真剣に考えず，とりあえず思いつきで十分です。すでにネタが溜まっていて，ある程度ストーリーが見えているなら，本筋を作るのもさして苦労はないでしょう。まったく思いつかないというのであれば，「誰が」という部分と，「なにをした」という部分を別々に考え，片っ端から組み合わせてみるのもよい方法です。

このように，本筋はデタラメに近いものであり，この段階で，このストーリーが面白いかどうかはあまり気にする必要はありません。本筋を5，6本用意し，以下の作業を進める過程で，行き詰まったり，面白くなりそうにないものを捨てていきます。

これからこの本筋を発展させていくわけですが，エピソードを考えたり，設定を練っていくとき，常にこの本筋に沿っているかどうかを確認しながら進めることが大切です。でないと，話に枝葉が増えてわかりにくいストーリーになったり，

大解剖!!これが、かめぽんだっ!!

かめぽん頭 なかなかかしこいぞ！小学生並だ！
かめぽん目 見えるぞ！
かめぽん口 いろいろ食べられるぞ！
かめぽん手
かめぽん腹 胃が弱ってるぞ！ストレスに気をつけるんだ！
かめぽん腕 曲がるぞ！力はそこそこだ！
かめぽん甲羅 けっこう硬いぞ！ほんとのかめ並だ！
かめぽん太もも
かめぽんつま先 立てるぞ！
かめぽんふくらはぎ なぜか足首のほうが太いぞ！

身長 174cm
体重 66kg
夜店で売っていたミドリガメを遺伝子改造したデミヒューマン

設定すべき項目 column2

○登場人物(特に主人公)の設定
・年齢，男女，外見，性格
・過去の人生(生まれてからいままで，どんな生き方をしてきたのか)
・現在の私的な生活(暇なときはなにをしているのか，常日頃どんな態度なのか，なにを考えているのか)
・この作品の中でなにを実現したいのか
・この作品の前後でどんな変化をとげるのか
○舞台・世界の設定
・場所と時代
・過去の場合は時代考証
・空想の世界の場合は矛盾のない世界観
○その他作品に応じた設定
・登場人物の特殊な能力など
・メカや武器の設定など

ちゃんとまとまらなくなるからです。

●3.テーマを考える

〝テーマとはなにか〟を真剣に考え出すとものすごく奥が深くなるので，今回はとりあえず簡単に，〝その作品を通じて，自分は視聴者になにがいいたいのか〟ということだと考えてください。具体的にいえば，主人公が本筋のように行動をすることや，その結果に対して賛成なのか，反対なのか，どう思うのかといったことを，自分なりの意見として持ってください。

たとえば，本筋が，
・お人好しの刑事が，悪徳企業を叩き潰す
であるなら，
・悪い奴らは徹底的に叩き潰してしまえばいいんだ
・ひとつの悪を潰しても，全体に影響はなく，世の中は変わらない
・ひとりの刑事には無理だが，多くの人が協力すれば可能なはずだ
といった感じで，人それぞれによって異なるでしょう。

つまりそれがあなたの個性であり，この作品の重要なポイントになります。逆のいい方をすれば，この時点で特に自分なりの意見がないようでしたら，完成してもテーマの弱い作品になってしまうでしょうから，その本筋は捨てたほうがよいと思います。もう少し自分の興味がある事柄についてネタを仕込み，本筋から立て直してください。

次に，テーマが明確になったら，本当に本筋が自分のテーマにふさわしいものか確認してください。

たとえば，本筋が，
・事故によりサイボーグとなった女が人間性を取り戻す
としていても，自分の考えやいいたいことが，
・人工知能といった機械も，心や人間性を持つかもしれない。そんな可能性を示唆したい

だったなら，もともと人間であるサイボーグが人間性を取り戻しても，このテーマを表現することはできないでしょう。テーマと本筋がちぐはぐです。

そこで，テーマを表現するためには，誰が，なにをするのがいちばん効果的かを考え，本筋を修正します。この例では，
・高度な知性を持った人工知能00086Xが，サイボーグの女との交流によって，人間性を得る
といったように修正するのがよいでしょう。

●4.あらすじを作る

さて次にあらすじを作ります。本筋をベースにして，テーマを盛り込み，そのテーマを表現するにはどういった話が効果的かを考えながら，少しだけ詳しく5，6文で書いてみましょう。この段階では，話の順番や具体的な内容まで書く必要はありません。すでに盛り込むことを決めているネタやエピソードもあるでしょうが，一部分だけ詳しく書くようなことはせず，あくまでも概略に留めます。

・人工知能00086Xは，極めて高度な知性を持つため、知性の低い人間をくだらない存在と考え，人間性を持つことを否定していた。そこに女性型ロボット(レイ)が現れ，その予測できない行動や可能性に驚かされる。実はレイはロボットではなく，脳は人間のままのサイボーグであった。人間性の重要性に気が付いた00086Xは，よりすばらしい人工知能として進化した。機械でも，人間性を持つことは可能で，それがよりよい機械を生むのかもしれない

・古代トルコの小国の王が，自分の存在に疑問を抱く。そして，国民に突然おふれを出し，小石を積み上げ，巨大な山を築かせる。それはあまりに無意味な行為に見え，民衆の怒りを買うが，王は自分がここに生きていた証拠を残したかったのだ。事実，2000年がたった現在，そこは観光地になり，王の名は語り継がれている。王の行為は，あまりに愚かだが，その思いは現代人も抱えているものだ

あらすじは，あくまでも指針とはいえ，この作品で表現すべきものが明確になります。なにか面白いアイデアが浮かんだとしても，このあらすじから外れているものは，ばっさりと切らなければいけません。そして，この内容をより強調するエピソードを考え，より効果的な構成を考えていけばよいのです。

なお，この段階ですでに面白ければいうことありませんが，ありがちでつまらないように思えることもあります。しかし，これから加えるエピソードや設定などが斬新なら，十分面白い作品になるので，まだあきらめる必要はありません。

●5.設定を用意する

あらすじができたら，シナリオを書く前に，登場人物や舞台となる世界の設定を考えましょう。机の前でうなっているだけではいけません。図書館やインターネットを利用して，細部にわたる専門的な知識を身につけるべきです。

登場人物の場合，細かい設定をすることで，その人物の何気ない動作にも差が出てきて，生活感，存在感が増し，視聴者は感情移入しやすくなります。舞台の設定でも，その世界が実在してい

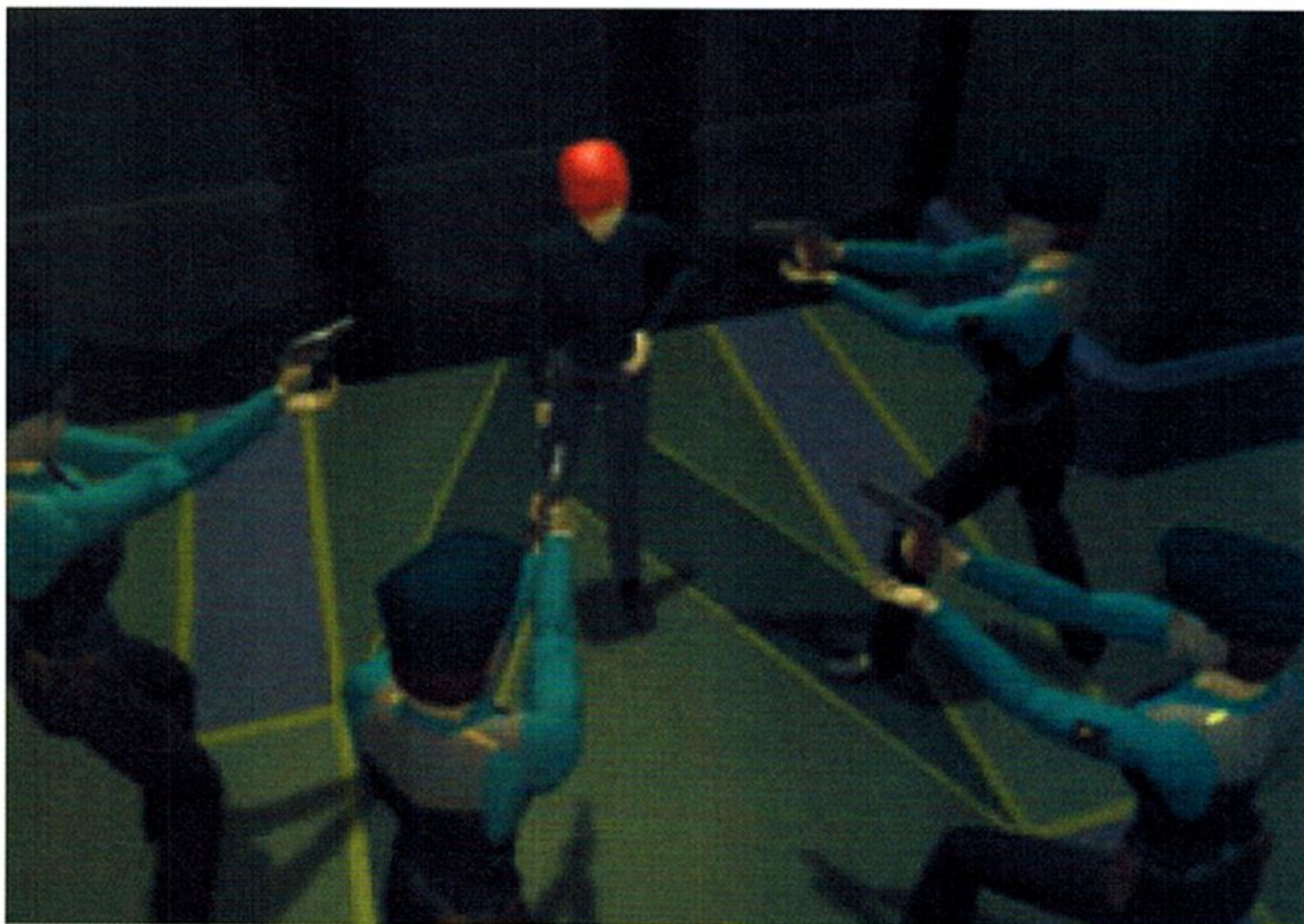
るかのようなリアリティが生まれます。

小道具など，細かいところに凝るのもよいでしょう。たとえば殺し屋の話を作っているとすれば，「世界の拳銃」といった本をチェックします。そして，その本で得た知識をもとに，シナリオで「H&KのP7か。なかなかいい銃だ。だが，そいつには8発弾倉と13弾倉がある。どうせなら13発にしておけ」といったセリフを設けると，全体のリアリティがグッと増します。また視聴者は，〝この作者は銃器に関してはかなり詳しい〟とだまされますので，別のところで多少非現実的な武器が出てきても，〝そんな武器もきっとあるんだろう〟と信じてくれます。さらに，主人公が手にしたP7が8発弾倉だったため，弾が足りずにピンチを招くというエピソードを思いつくというように，新しいアイデアにもつながるものです。

ただ，設定に凝ったからといって，それをすべて盛り込もうとするのはいけません。解説的なカットが多くなったり，ストーリーに枝葉が増えて，無理が生じます。

●6.エピソードを考える

あらすじにある個々のエピソード，つまり主人公が本筋にある行動を取るようになった理由や，その目的を勝ち得た方法などを具体的に決めていきます。いままで温めてきたネタや設定といったアイデアを注ぎ込みましょう。クライマックスなど重要なポイントには特にオリジナリティや意外性が求められます。逆に重要度の低いところは簡潔に描けるエピソードのほうがよいでしょう。

少し長い作品の場合，目的を達成するまでの障害の数を増やします。つまり，主人公が目的に向かって動き出すとそれに対する障害が現れる。その障害をクリアすると，さらに次の障害が出てくるというのがストーリーの基本パターンです。

この障害やクリアするアイデアが足りないときは，友人たちに助けを求めるのもよい方法です。もしある程度の頭数があるなら，2チームに分かれ，片方のチームは意地悪な障害ばかりを考え，もう片方がその障害をクリアする方法を考えるというやり方もあります。

逆にエピソードがいろいろ考えられる場合の取捨選択は，常にあらすじやテーマをふまえて考えます。アイデアとして面白くても，テーマやあらすじにあっていないものは使えません。また，実際に自分がCGとして制作することを考え，制作に無理がないように留意することも必要です。

●7.シナリオを書く

各エピソードも決まったら，あとはあなたの頭の中にあるストーリーを，シナリオという形で書いていきます。シナリオは，あまり決まった書き方などありません。特に個人制作の場合，自分さえわかれば，基本的にどんな書き方でも結構です。共同制作なら，そのシナリオを読む仲間がわかりさえすればいいのです。

最初の段階では，各シーンごとに，どういったことが起こるかを詳しく書いた程度でもよいでしょう。推敲が進んで完成に近づけば，絵コンテの作業を軽減するためにも，全部のセリフ，全部の

シナリオサンプル

column3

【時】3，4世紀頃(まだ鉄砲がないころ)
【場所】中国の奥地の小さな村
(南部の広西壮地区，桂林の璃江の近く)
【世界観】チャイニーズ・ゴースト・ストーリー，犬夜叉魔物がいたり，仙人がいたりする世界

●オープニング

OPN01■のどかな風景(ロング)，いかにも中国らしい山水画のような山
ティルトダウンすると，薬師がひとり歩いている。

OPN02■歩いている薬師の下半身のトラック。村の境界を示す道祖神の小さな祠がある。薬師，気がついて足を止める。

OPN03■かがんで道祖神にお祈りする薬師。フルに近いロング。明るく爽やかな感じ。背景はほとんど真っ白，薬師は逆光でほとんど影。背後の木々がそよいでいる。
薬師，最初軽く手を合わせ，それからさらに深く頭を下げる。

●タイトル

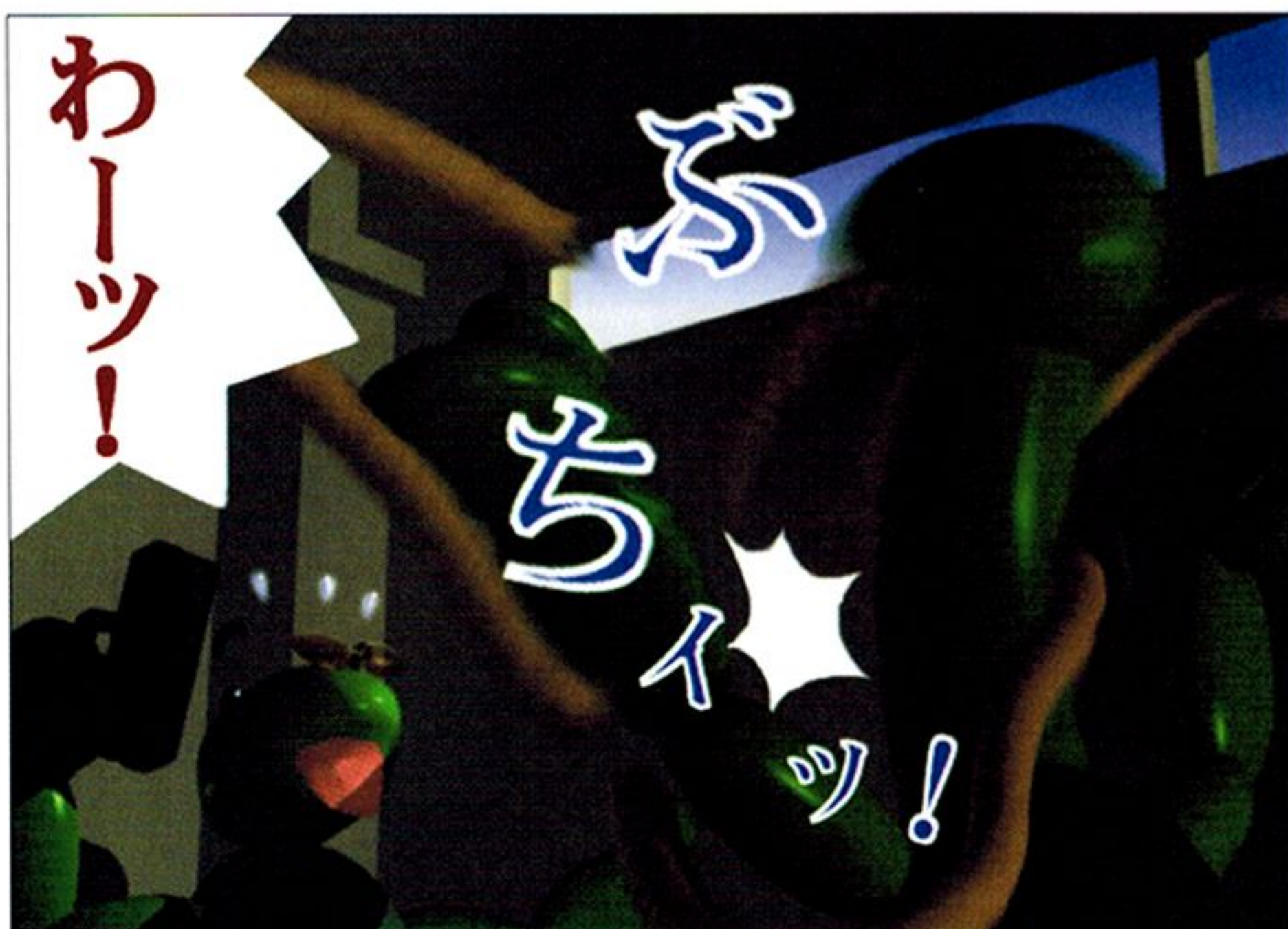
TIT01■メインタイトル「対魔薬師」
急に動的なカット。「対」「魔」「薬」，「師」の文字が，カッカッと飛んで現れ，最後に背景に梵字で書かれた魔法陣が光る

IT02■サブタイトル
第一話　薬師戦紅炎魔王(漢文調，レ点など入れる)

●魔王登場

MAO01■夕闇が迫り，霧が出て，少しおどろおどろしい感じ。
わりと立派な屋敷の門
ティルトダウン
門に張り紙

MAO02■怪しげな張り紙のアップ。
中国語でなんか書いてある
「今天晩上 我奪弥的娘　紅炎魔王」
テロップ：「今夜娘をもらいに来る紅炎魔王」
薬師がフレームイン。張り紙を見たあと，門を開ける。

MAO03■娘のアップ。恐ろしさに顔が引きつる。フラッシュカットを使って急にテンポを変える。
娘「キャー！」
(以下省略)

アクションを事細かに記し，カットの切れ目(カット名)，カメラワークまで指定すれば完璧です(シナリオサンプル参照)。

その場合の注意としては，心理描写など，映像化できないことは書いてはいけません。たとえば，

・A子，B男が立ち去る様子をぼんやりと見ている。顔は無表情だが，B男に対する怒りが沸々と湧いてくる

とシナリオに書かれても，映像ではA子がぼんやりしているだけで，B男に対する怒りは表現できません。ですからここは，

・A子，B男が立ち去る様子をぼんやりと見ている
・A子，顔は無表情だが，手に持っていた2人の写真をゆっくり，何度も破る

というふうに書く必要があります。

●8.推敲する

ひととおりストーリーができたら完成というのは甘く，何度も推敲を重ねなければいけません。実際の作業としては，「7.シナリオを書く」とループするかたちになります。

まずは，明らかな矛盾や無理がないかチェックします。そんなことはいわれなくてもわかっていると思うかもしれませんが，書いた本人は意外と気づかないことが多いのです。そこで，友人などに読んでもらい，変なところ，意味がわからないところを指摘してもらうとよいでしょう。もっとも，読み手側に問題がある場合も少なくないので，できれば複数の方に頼みましょう。

そのとき，〝ありきたり〟といわれれば，指摘された部分のアイデアは再考してみるべきです。〝いや，これ以上のアイデアはない〟と思った瞬間に思考は止まってしまいます。〝もっとよい方法が必ずあるはずだ〟とあきらめずに考えてください。単に指摘された部分だけに注目せず，その前後ごとがっさり変えるとか，順番を入れ替えるとか，別の視点で見るとまた新しいアイデアが出てくるものです。

そして，推敲においてもっとも重要なことは削ることです。ここまで各ステップをちゃんとこなしてきたなら，あなたの脚本にはたくさんの設定やアイデアが盛り込まれ，とうてい10分では描けないような超大作になっていると思います。しかし，最初に述べたように，初心者がそんなに長い作品など作れっこないのですから，2分程度に削らないといけません。せっかくの設定やアイデアの大半を，断固たる意志を持って捨て去るのです。

これは辛い作業であり，初心者にはなかなかできません。だからといって削り切らなければ，結局作品は未完に終わってしまいます。そこで，これはパイロットフィルムなんだと自分をだましてみましょう。つまり，いずれ30分のOVAが作られるが，まずはスポンサーにこの脚本の面白さがわかるようなダイジェスト版を作って見せるんだと考えるわけです。クライマックスだけを映像化するか，予告編のように，特に面白いところ，ストーリー上絶対に必要なカットだけを作るぐらいに考えてください。

視聴者という者は，足りない部分は結構適当に想像してくれるものです。だから，お決まりの展開のほとんどすべては省略が可能です。たとえば，ロマのフ比嘉さんの傑作「SANDSTORM」では，〝変身能力を身につけた主人公が，ある事件の黒幕たちを次々と抹殺していきます。そして最後に，最大の黒幕の本拠地に単身で潜入します。しかしそこは元上司の警察官たちが警備しており，主人公は追いつめられ，周りを囲まれてしまいます。〟という部分がバッサリと省略され，最初の1カット目ですでに警察官に囲まれています。どうやってそこまで潜入したかとか，ほかの黒幕たちがどうやってやられたとかいうのは，作品中に描かれていません。それどころか，この黒幕たちがどんな悪事を働いたかさえ視聴者の想像に任せています。省略しようと思えば，ここまでできるというよい見本だといえるでしょう。

以上，ひとつのストーリーの作成法を解説しました。この方法は，最初に本筋やテーマをはっきりさせるので，比較的シンプルでわかりやすいストーリーができる半面，アイデアが不足するとありきたりになるという欠点があります。ともあれ，ストーリーが思いつかない方は，まずはこの方法で挑戦してみてください。

ストーリーの作成法は，もちろんこれ以外の方法もいろいろあります。先にBGMを決めて，その曲を聴きながらイメージを膨らませたり，既存のストーリー(昔話など)をアレンジする方法などもあるでしょう。いろいろ試して，最終的には自分にあった方法を見つけてください。

第3章:ストーリーをよりよくするポイント

この章では，過去多くの自主制作CGアニメを見てきた経験から得られた，よくありがちなミス，または改良のヒントなどをまとめてみました。企画や推敲のときの参考にしてください。

●視聴者の好み

自主制作では，自分の作りたいものを作るというのが原則ですが，現実問題として，誰も見てくれないような，嫌がられるような作品を作ってもむなしいだけです。視聴者がなんのために作品を見るかといえば，気晴らしがしたい，楽しくなりたい，幸せな気分になりたいからです。決して陰鬱になりたいとは思っていません。

にもかかわらず，ときどき登場人物がみんな死んでしまったり，やたらグロテスクな作品を作る方がいます。劇場映画などでは少ないパターンだから，オリジナリティがあると考えるのは少し軽率です。誰も思いつかないのではなく，受けないので避けられているという見方が正しいでしょう。

なにも勧善懲悪を勧めるつもりはありませんが，奇をてらった作品を作る場合，それなりの必然性があるべきです。

●〝謎〟の提示

視聴者をストーリーに引き込むためには，視聴者が興味を持つような〝謎〟を提示するのが有効です。たとえば，子供の誘拐を題材にした作品の場合，〝この子はどうやって救出されるんだろう?〟というのがこのストーリーの最大の謎になります。また，襲ってくる敵をあえて説明せず，〝奴らは何者で，なぜ襲ってくるのか?〟という謎を作ることもできます。

この種の謎は，ストーリーのできるだけ早い段階で提示するべきです。でないと，視聴者は作品に対して関心を持つ前に退屈してしまいます。たとえば，上記の誘拐ネタの場合，テーマが親子の愛情で，誘拐される前の親子のすれ違いの日常を描く必要があっても，冒頭からそんなシーンを長々と見せるのは考えものです。そんな場合，いきなり子供が誘拐されるシーンから入って視聴者の興味を引きつけてから，回想シーンで誘拐される前の様子を描くという方法もあります。

また，提示した謎がすべて解けてしまうと，作品に対する視聴者の関心は急激に薄れてしまいま

す。ですから，謎が解決すればさっさと終わるということも重要なことで，ラストでうだうだしていると，作品の評価は下がる一方です。

●意外性と伏線

ストーリーにおいて，クライマックスで主人公が最大の障害をいかにしてクリアするかが最大の見所です。この部分で視聴者に〝なるほど！　面白い〞と思わせるためには，従来にはないオリジナリティと，事前に予想できない意外性が不可欠です。

しかし，いくら意外性があっても，唐突で脈絡がないのはいけません。たとえば〝か弱い女性が屈強な男に拳銃を突きつけられる。誰もいないところで，どうやってこの男をやっつけるか？〞というクライマックスで，いきなりどこからともなく犬がやってきて，かみついたという展開では，視聴者は怒ってしまいます。

ところがこんな唐突な展開も，事前に伏線を見せておくことで十分使えるネタになります。

- 主人公の女性は犬を飼っており，この犬は革製品を見るとかみつく癖がある
- この犬と散歩中に女性は車で連れ去られ，犬だけ置き去りになっていた
- 男が拳銃を持つ前に，革の手袋をはめる

となれば，犬が臭いを頼りに主人公のところまできて，いつもの癖で手袋にかみついたと視聴者は納得します。

ただ，伏線部分で犬の癖をやたら強調していたり，置き去りになった犬が車に向かって走っていったり，手袋をはめるとき，わざわざ「これは革製品だ」などという説明的なセリフがあると，視聴者もこの展開が予想でき，せっかくの意外性が台なしになってしまいます。ですから，伏線はいかに気づかれないように，さりげなく入れるかということが，大きなポイントになります。あんまりさりげなく，視聴者の記憶に残らないのもいけませんから，このへんのさじ加減が腕の見せどころです。

●感情移入

視聴者は，登場人物の誰か(多くは主人公)に感情移入し，その人物を応援しながら作品を見ます。たとえば，逃げる容疑者と追いかける刑事を描く場合，容疑者の立場か刑事の立場か，どちらか一方を応援します。

ですから制作者も，視聴者が特定の人物に感情移入するように，意図的にコントロールしないといけません。それが容疑者の立場なら，この容疑はぬれぎぬだとか，相手が悪徳刑事だとか，この仕事さえ終われば足を洗い平和な余生が送れるとか，容疑者に同情し，応援したくなるようなエピソードが必要になります。

これを怠る，あるいは下手に両方を対等に描いてしまうと，視聴者はどちら側にも感情移入できません。両者を他人ごとのように客観的に見てしまい，野球に例えるならファンではないチームどうしの試合のように，盛り上がらないのです。

●セリフ

セリフの量は可能な限り減らしましょう。一語ずつ省略できないかよく検討してみてください。特にセリフが音声ではなくテロップの場合，テロップを読むのに忙しすぎて，画面を見ている暇がなくなっては，映像作品として本末転倒です。徹底的に削ってください。

また，画面で見ればわかるようなことを，わざわざセリフにするのは無駄です。これは，ナレーションでも同じことがいえます。

たとえば，

OPN01■砂漠のような世界に白い塔がある。
男が塔のほうへフラフラと歩いている。
ナレーション：夢を見た。私は白い塔に向かって歩いていた……。

とすると，後半が重複しています。

また，書き言葉と話し言葉の違いにも注意が必要です。普通にしゃべっているときは，結構いい加減な日本語になっているものです。あまりに理路整然としゃべるとリアリティがありません。また，複数の人が会話しているときは，多くの人が相手の発言が完全に終わるのを待っていません。どんどん，次のセリフを割り込ませるようにするほうがよいでしょう。

●視聴者の目で考える

制作者は，自分の知っていることは，視聴者もわかっているかのような勘違いをすることがあります。

たとえば，作品の冒頭で，

OPN01■のどかな風景(ロング)，いかにも中国らしい山水画のような山。
ティルトダウンすると，道を旅人やら，町人やらが歩いている。
主人公も，のんびり歩いている。
OPN02■主人公，ふるさとの町へ入っていく。

とあったとしましょう。

この第1カットの場合，制作者は主人公を知っていますが，視聴者は道を歩いている人達の誰が主人公なのかさっぱりわかりません。主人公とは反対側に歩いていった美しい町人の娘のほうに注目しているかもしれません。同様に第2カットでも，これでは主人公が，見知らぬ町へ着いたのか，自分の町に帰ってきたのか，視聴者にはわかりません。第3カット以降に，それを示すセリフなどを入れてやる必要があります。

またサイレント映画のようにセリフを声ではなく文字で出す場合，制作者にとっては，各々誰のセリフか明白でも，視聴者には結構わかりづらいものです。下手に取り違えられると，ストーリーが全然わからなくなることもあります。誤解を招くおそれがないか，視聴者の目でチェックしてみてください。

●秘すれば花

能を完成させた観阿弥・世阿弥がその極意を〝秘すれば花〞と述べているそうです。簡単にいえば，〝重要なことはストレートに出さないほうがよい〞といった意味で，これはなかなか応用がききます。

たとえば，誰かが死んで主人公が悲しんでいる様子を描く場合，主人公がわんわんと泣き，「悲しい，悲しいよ」と叫ぶと，視聴者は白けてしまいます。むしろ，悲しいのをじっとこらえているほうがはるかに悲しみが伝わるのです。

同様に，主人公がテーマそのものを語ったり，テロップででかでかと表示するなどというのはもってのほかです。視聴者にテーマをあまり押しつけると賛同を得るどころか，反感を買いかねません。

そのほか，重要なセリフやカットをあえて省略したり，はっきり出さないという方法もあります。復讐の相手をやっと追いつめ，銃口を向けて，とどめを刺す瞬間，あえてロングにし，暗闇の中で銃口の炎だけが一瞬光るなんてのもしぶい演出です。

この手の省略のテクニックを使う場合，前後からそのセリフやカットが推測できる必要があります。しかし，あえて十分な情報を提示しないことで，視聴者に考えてもらうという応用テクニックもあります。たとえば塩竈氏の「ある暑い日に」では，地球の温暖化によって水没してしまった世界が舞台となっていますが，終始屋内のカットだけで，水没した都市の風景は最後まで描かれません。それによって視聴者は〝どのくらい水面が上昇しているのだろう。どうしてそんなことになったんだろう〞と考えさせられるわけです。

以上のようにストーリー制作上の注意点などを述べてきましたが，これらすべてを満たしている必要はありません。また，例外はつきものですから，意図的に原則に反してみるのもよいでしょう。

今回のストーリー作成法を読んだからといって，その日からスラスラとストーリーができるわけではありません。しかし体力トレーニングと同じように，持続が大切です。寝てもさめてもストーリーを練り，なにを見ても聞いても，〝これはネタになるんじゃないか〞と考えましょう。数カ月後には身についてくるはずです。

次回のCGAコンテストの締め切りは2000年2月14日です。ぜひ成果を拝見したいと思います。

特別企画 **Welcome to Immersive Space.**

"Immersive"という単語は日本語にはしづらい。「浸る」というのがニュアンス的にいちばん近い意味だろうか。没頭しきる感覚，その世界の中に浸り込んでしまえるような状況を称してImersiveと表現する。
そこでは単に画像情報としての3Dというより，それが引き起こす「感覚」のほうが重要視される。疑似体験の「疑似」の部分をどれだけ色の薄いものにできるかが重要だ。自由に生成できる3D映像というものはすでに我々は手に入れているのだが，活用例はまだまだぱっとしない。3Dであることの必然性が感じられるものが少ないからだろう。
技術的にはバーチャルリアリティという絵に描いた餅が次第に現実的な話になってきて，あと一歩さえ乗り越えれば，映像メディア全体が革命を起こす……はずなのだが，なかなか市場が追いついてこない。現状では，パソコンゲームが時代の最先端の位置につけているのだが，さらにもう一歩向こうへ踏み出さなければならないのだ。

立体空間の誘惑

中野修一　Nakano Shuichi

Dreamcastや PS2（仮称）が発表されるたびに残念に思っていることがある。表示系がテレビしか想定されていないことだ。PS2などは高解像度の表示，DreamcastはVGA出力を持っているとはいうものの，十分とはいいがたい。どんなに優れた表示性能を持っていても，それが低解像度で固定フレームレートであれば，その機能の実現に費やしたコストほどの効果は得られないものだ。

3D性能もPlayStationクラスなら，ちょっとまともな表示にするだけで段違いの，そしてそれ以上はあまり求められていないくらいのレベルに達すると信じている。パースコレクトとVRAM拡大，可能ならアンチエイリアスといった程度で十分だろう。家庭用テレビを前提としている限り，それ以上投資しても報われる部分は少なそうだ。実際のところ，家庭用ゲーム機の限界はテレビ出力を前提としているところにこそある。

個人的にブレイクスルーとなるのは3D表示しかないと思っている。なのに，どこも3D表示をやってこない。私にいわせるとこんなおかしなことはない。店頭でオメガブラストのデモを初めて見たときの感想は「なんで立体じゃないんだろう？」というものだった。立体表示デバイスがないからというのが正解なのだろうが，なにか凄くもったいないことがされているような気がするのだ。

君は立体映像を知っているか？

よい立体映像に出合うと人生観が変わる。

立体映像と聞いて，赤青メガネや目を凝らしての裸眼立体視しか思いつかない人は，おそらく「ふ〜ん」と思うだけだろう。ちょっと浮き出て見える絵になったところで，そんなに大事だとは想像つくものではない。ヘッドマウントディスプレイとかいっても，遠く離れたところに小さく映っているだけという印象しか持っていないだろう。これが，数100万円クラスのデバイスを使うととんでもないことになる。視界に広がる風景はまさに「リアリティ」を持つ。立体映像や，揶揄の対象でしかなかったバーチャルリアリティという言葉の真の意味を知った瞬間だった。オリンパスの業務用システムが300万円と聞いて「個人売りはしないんですか？」と思わず聞いてしまいそうになったことがある。5年くらい前の話だ。

そこで行われていたのはポリゴンで作られた街のフライスルーデモだった。飛行する視点で欧風の街並みの中を飛び回る。路地を抜け，アーチをくぐり，塔の先端まで吹き上がる。眼前で展開されるリアルタイム映像がもたらすのは「飛行感覚」だ。空に向かって舞い上がり，足元に広がる夕暮れの街並みを見下ろすとき，空を飛ぶというのはまさにこういう感覚なのだと気づく。あんなふうに空を飛んだことなどありはしないが，だからこそバーチャルなリアリティというものの価値を体感できる。現状のリアルタイム3DCG技術なら，飛行している映像を作り出すことは簡単にできる。しかし，飛行感覚を作り出すにはディスプレイ上に表示させていてはダメなのだ。

そして，時は流れ，リアルタイム3D映像自体はご家庭で当たり前に生成されるようになった。しかし，立体映像のために3Dを使っているわけではなく，映像生成自体のために3Dを基にしたほうが都合がいいというのが現在の流れだ。ゲームは映画を目指す，とおエラいさんがいってたりもするが，映画並みの画質であるということにどれくらいの意味があるのだろうか。映画の力の大半はスクリーンサイズによる刺激の量的なものであって，画質などによる影響は微々たるものだ。まったく世間の人は欲がなさすぎて，これでは技術がなかなか進歩していかない。

写真1　フィードバック付きデータグローブであるサイバーグラスプは映像を「触れるメディア」として提供できる（以下，産業用バーチャルリアリティ展より）

にもかかわらず，技術的な背景は揃いつつある。リアルタイム映像生成，ヘッドマウントディスプレイ。フォースフィードバックコントローラや3Dサウンド技術など，ほぼ聴牌状態なのがわかる。

なければ作る，というわけではないが，すべての映像メディアは立体に移行すべき時期にきている，とぶち上げたうえでパソコンによる立体表現について考えていくことにしたい。

EyeScream

最近PCで脚光を浴びている3D表示方式といえば，METABYTEのEYE SCREAMによるものだろう。液晶シャッター方式を使ったものだが，Direct3DとVoodoo Glideドライバレベルで動作するため一般的な3Dゲームの大半がそのまま立体表示されるという画期的なものだった。こ

写真2　データ取り込みもレーザーで何度かなでるだけ，スキャンできるようなものが登場している

写真3　ELSA ERAZOR Ⅲと3D表示システム

の方式はDirect3Dに取り入れられたようで，最近では各社の3Dビデオカードで同種の方式を使用した製品が現れてきている。ディスプレイを120Hzで動作させることで，液晶シャッター特有のちらつきはほとんど解消されている。手軽なうえ，決して悪いものではない。これからの時代，120Hzの出ないディスプレイは買い換えたほうがいい，といい切ってしまいたいくらいの技術だ。ASUSTeKのTNT2カードやELSAのERAZOR III，WINNER IIなどが対応している。

さて，この方式の最大の欠点はメガネ以外に，そこそこ強力なディスプレイを必要とするということだ。安めのディスプレイはまずアウト。液晶ディスプレイもだめだ。もっとも世の中には，強電誘導体液晶で120ヘルツで動作する液晶板というのも存在するのだが。

偏光デバイスの可能性

立体の基本は左右の目に別々の映像を与えることである。映像ソースを2つ用意するのが万全だが，世の中そうもいかない。映像機器を2セットずつ使わないとできないようでは敷居が高い。そこで，ひとつのソースから2つの情報を取り出したいと思うわけだ。液晶シャッターも奇数フィールドと偶数フィールド，要するに，奇数ライン目と偶数ライン目で画像を変えていたわけだ。取り出す方法が液晶シャッターだっただけの話だ。

液晶シャッター以外の方法で同じ情報を取り出そうというものもある。それが偏光グラスである。光は横波でいろんな方向を向いていると考えていいわけだが，特殊なフィルタを通すことによって，ある向きの光の波だけを取り出すということができる。これが偏光フィルタというものだ。

縦に置かれれば縦の波だけを通す。横に置かれれば横の波だけを通す。これを細かくコントロールしてやれば面白いことができる。液晶パネルの奇数ライン目のドット位置には横方向の偏光フィルタが貼ってある。偶数ライン目には縦方向の偏光フィルタが貼ってある。この液晶ディスプレイを横方向の偏光フィルタ越しに眺めると，奇数ライン目の映像しか見えない。偶数ライン目から出る光は向きが違うので偏光フィルタを通過できないのだ。同様な処理を左右の目に対してやれば，同じ画面を見ていても右目と左目には別々の映像が入ってくる。写真4はそのような液晶ディスプレイによる立体映像を撮影したものだ。

写真5はこの原理でさらに半球上のドーム内に2基の液晶プロジェクタからの映像を投影して見るという装置だ。視野角はきわめて広くて快適だ。このドームは「比較的低コスト」で実現できるという。右を向けば右の画像が見えるというのは非常に素晴らしい。Zガンダムの世界は案外近いのかもしれない。

ヘッドマウントディスプレイの憂鬱

立体映像を得る手段にはいくつかあるが，Immersiveという意味ではヘッドマウントディスプレイが最右翼だ。前述のとおり，最近は液晶シャッター方式も120fpsディスプレイを使用することにより実用的な画像を生成してはいるが，家庭用テレビとの組み合わせは厳しいものがある。なお，世の中には120Hz表示のテレビというのもあるにはあるが，決して一般的ではない。

写真4　FLD液晶で120Hz動作可能な液晶ディスプレイ

いずれにせよ，眼前に広がる光景というものを実現するにはのぞき込むテレビタイプではダメだ。

液晶ディスプレイの進化を見ていると，ヘッドマウントディスプレイなど，もっと簡単にできそうなものではないかという気もするのだが，なかなかそうもいかない。

液晶の違いもあるのだろうがヘッドマウントディスプレイはまだまだ高すぎる。秋葉原で1万円で売られている液晶テレビを見るたびに，「これ2個でいけるかな……」と思うのだが，光学系をちゃんと作らないと目の焦点があわないのはわかりきっている。ドット密度なども低いので，装置自体も大掛かりになりそうでもある（一度は試作してみたほうがいいのかもしれない）。

現行製品の問題点を見てみよう。

映画などを鑑賞するために作られているというのがまずいけない。液晶の解像度というのはさほど高いものではない。それが眼前にあるように拡大されると画素が大きく目だって見えるのは当然だ。結局「画質が悪い」ということになってしまう。そういう「鑑賞用」ではちょっとぐらい遠くても綺麗な絵のほうが喜ばれるらしい。現状の製品はすべてその系統のものだ。ゲーム用途であればドットが見えても多くの人はあまり気にしないだろう。相手はついこないだまでスーファミなどを使っていた人々なのだから。そういう用途の迫力重視設計の製品を待ちわびているのだが，なかなか現れない。EyeTrekも最近出たワイド画面対応版に切り替えようか……。

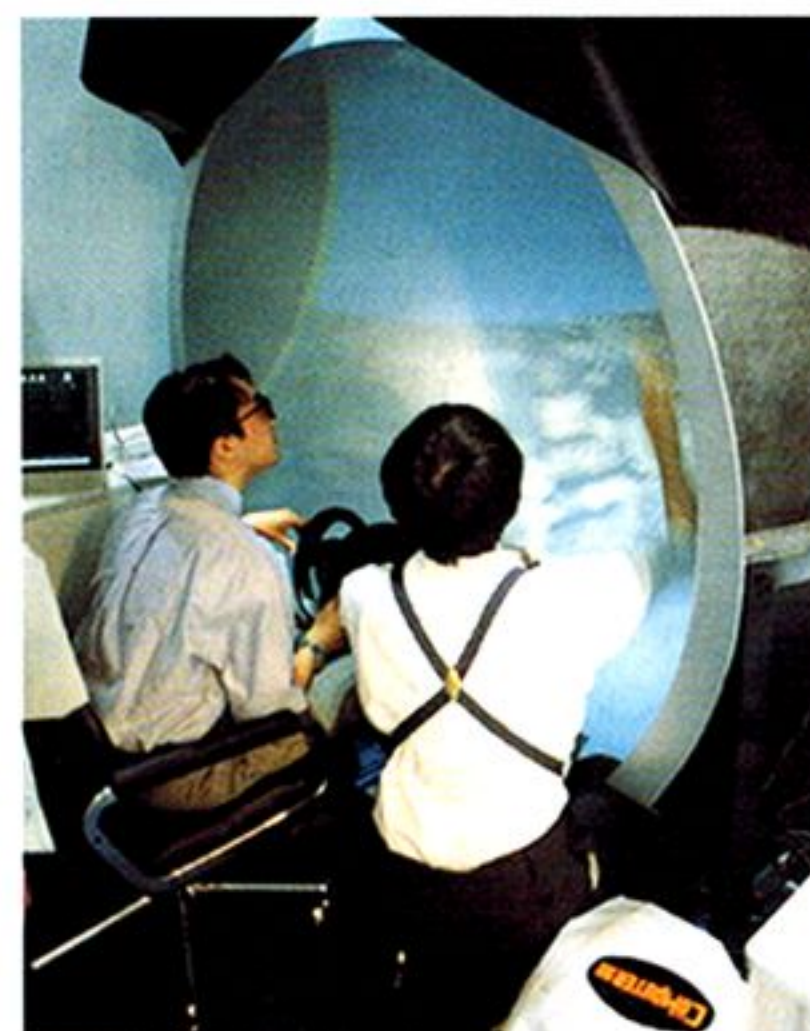



写真5　半球内に2つのプロジェクタからの画像を投影するシステム。都市開発用に作られたもの

ここでちょっと目的にかなうものはどんなものかを考えてみる。

・視野角

絵は歪んでもいいから，視野のできるだけ広い部分に映像が必要。

・密閉型

安全を考えてか，開放型多いのもいけない。EyeTrekあたりは歩きながら見るための装備も揃っているが，それはきわめて例外的な用途だと思う。液晶を透かして周りが見えるとかいう製品の考え方はわからないではないが，はっきりいって，オープンタイプのものについていても無意味。

グラストロンなどはヘッドマウントでありながら，のぞき込むタイプだ。見づらいこともさることながら，画面外の不要な情報が多すぎてImmerseできない。密閉型は必須条件だ。ダイノバイザーくらいのがいいんだが，EyeTrekの開放部分をスモークアクリルでふさぐくらいのものでも妥当だろう。

・2系統入力

液晶は2つ使っても映像はひとつしか流れない。右目と左目で別々の番組をみたい人だっていると思うのだが。ビデオ入力2つさえあれば，あとはPlayStatin2台を対戦ケーブルで同期させて……とかも不可能ではないだろうに。

立体映像の生成

出力デバイスがないので，立体ソースもない。という悪循環はリアルタイムイメージ生成ができ

写真6　偏光フィルタを使った立体ディスプレイの画像を偏光フィルタ（CPL）で撮影。立体に見える？

ればひとまず解消される。3D表示が低コストでできるようになった現在こそ，立体映像は注目されてしかるべきなのだ。

しかし，もう一歩踏み込んで2次元映像を自動的に3次元映像にしようという動きもある。古くはパイオニアの「立体君」，最新のものではサンヨーが開発している2D/3Dコンバータがある。立体君は動作を見ると画面の動きから距離を判定して左右の画像を作っているように思われる。ときに前後関係を間違えたりするが，なんとなく立体ぽくはなる。サンヨーの装置は動きなどがなくても2D画像を3D画像に変換できる。それもかなり自然にできているようで，かなり驚嘆すべき技術だ。自動処理であれば破綻することもあるのだろうが，1枚絵（ミレーの落ち穂拾いとか）がそれらしく立体化されるのには驚かされる。ほかにも現在販売されているものだと，同種のものにレッツコーポレーションのRealEyes3Dなどがある（http://lets-co.co.jp/）。ただの2D画像に耐えられなくなったら，このようなすべてを3D化する仕組みも重要になるのだろうか。

写真7　これがパイオニアの立体君（レアかも）

写真8　サンヨーの3Dプロセッサ。なんでも立体にする

桑野氏に依頼された3D表示デバイス

NTSC信号は横方向の解像度（水平解像度）と縦方向の解像度（垂直解像度）では大きな違いがある。縦方向が走査線数により物理的に制限されているのに対し（最大で512本程度），水平方向は信号の詰め込み方次第で解像度が上がるのだ。水平解像度400本というディスプレイは水平方向に並んだ垂直線を400本識別できるという解像度を意味する。640×480ドット表示のデバイスで独立して引ける垂直線の数は320本だ。若干横方向のほうが余裕があることがわかる。機材さえよければさらに向上する可能性もある。

立体表示でいちばんメジャーな液晶シャッター方式は，インタレースモードの奇数番目と偶数番目の走査線を左右の目に割り当てる。要するに，縦方向の解像度が半分になってしまう。横方向の解像度低下はかなりアナログ的な変化であるのに対し（オートリニア補間？），垂直方向はデジタル

Column 疑似3DのDirect3Dによる実装

動物の目ってのはうまくしたもんで，2個で一対，それによって対象との距離を測定している。三角測量の原理だ。それを脳の中で無意識にやってしまって，リアルタイムで距離感としてフィードバックされる。まあ直接的な距離をいい当てるのには少々経験と訓練が必要かもしれないが，2つの異なる距離にあるオブジェクトの相対位置は，瞬時に判断できる。

たとえば図1見てもらいたい。丸が2つずつ，片方は寄っているが片方は離れている。これだけでは特に意味のある絵とも思えないが，この2つの絵を別々の目，右の絵を右目で，左の絵を左目，あるいはその逆で見てみよう。ひと昔前に流行った立体裸眼視とかステレオグラムと同じだ。

それぞれの絵を同じ側の目で見た場合，つまり平行法では左の丸が，逆の交差法では右の丸が飛び出して見えるはずだ。そう感じてしまうのは，「え～と，右目に写った像は左目のより離れてるから……」なんて思考が働いているわけではなく，まさしく感覚的な，膝をハンマーで叩くとぴょこんと反応してしまう「かっけ」のテストのような，反射神経みたいなものだ。このように画像処理をハードウェアでやってしまう「目」だが，脳の介入を許さないがために，さっきの丸のように，左右別の像が入力されているにも関わらず（脳では分かっているのに），同じ像を写しているものだとして処理し，結果として間違った距離感を出力してしまう。反対にいえば，ここに2次元だけで3次元的に見せてしまえるスキがある。

Direct3Dでそういった描画を行わせるにはどうしたらよいか，ということに焦点を当てていく（ここでの説明はDirect3D RMに関して）。幸いDirect3Dにはカメラに当たる「ビューポート」という概念があり，そのカメラは「フレーム」に載せることで，好きなところから好きな向きで撮影できる。要はそのビューポートを2つ作って，目の間隔だけ離せばよい。あとは描画時に，たとえば画面モードが640×480ならば，片方のカメラを左半分の320×480に，もう片方を右半分の320×480に表示するようにすればよい。具体的には，ビューの作成時に，

```
lpD3DRM->CreateViewport (lpDev,lpCamera,0,0,320,480,&lpD3DRMViewport);
```

とすれば，lpD3DRMViewportは画面左半分に描画するビューポートのインタフェイスとなる。右半分なら，

```
lpD3DRM->CreateViewport (lpDev,lpCamera,320,0,320,480,&lpD3DRMViewport);
```

だ。また，ビューポートを作成したあとでも，

```
lpD3DRMViewport->Configure (0,0,320,480);
```

とすることで，左半分を描画領域に変更することができる。

ただし，これだけではいまいちうまくない。というのは，2つのビューポートが，つまり左右の視線が平行なのだ。これでは視点の定まらないうつろな目になってしまう。ここはカメラフレームに適当に角度をつけて，視点を設定してやろう。ただし，ここは回転角度を指定するのではなく，たとえばカメラの間隔を2.0，視点を10.0先にするならば，

```
lpCamera->SetOrientation (lpRef,
-D3DVAL (1.0) ,D3DVAL (0.0) ,D3DVAL (10.0),
D3DVAL (0.0) ,D3DVAL (1.0) ,D3DVAL (0.0));
```

として軸を指定したほうが明確でいいだろう。2行目の3つのパラメータは，カメラフレームに設定するz軸（奥行き）を，3行目はy軸（上方向）をそれぞれlpRefフレームのベクトルで指定している（例は右目のカメラ）。

ただし，右目のカメラは画面左半分に，左目は右半分に描画，つまり交差法にするべきだろう。なぜなら，15インチのモニタでも，左右の画像の間隔は15センチ以上，反対ではよほどの平目顔の人でない限り，平行法どころかロンパリ法になってしまう（そういえば，ロンパリの語源が「片目がロンドン，片目がパリを向いている」と教えてくれたのは，高校の数学の先生だったなぁ）。（菊地功）

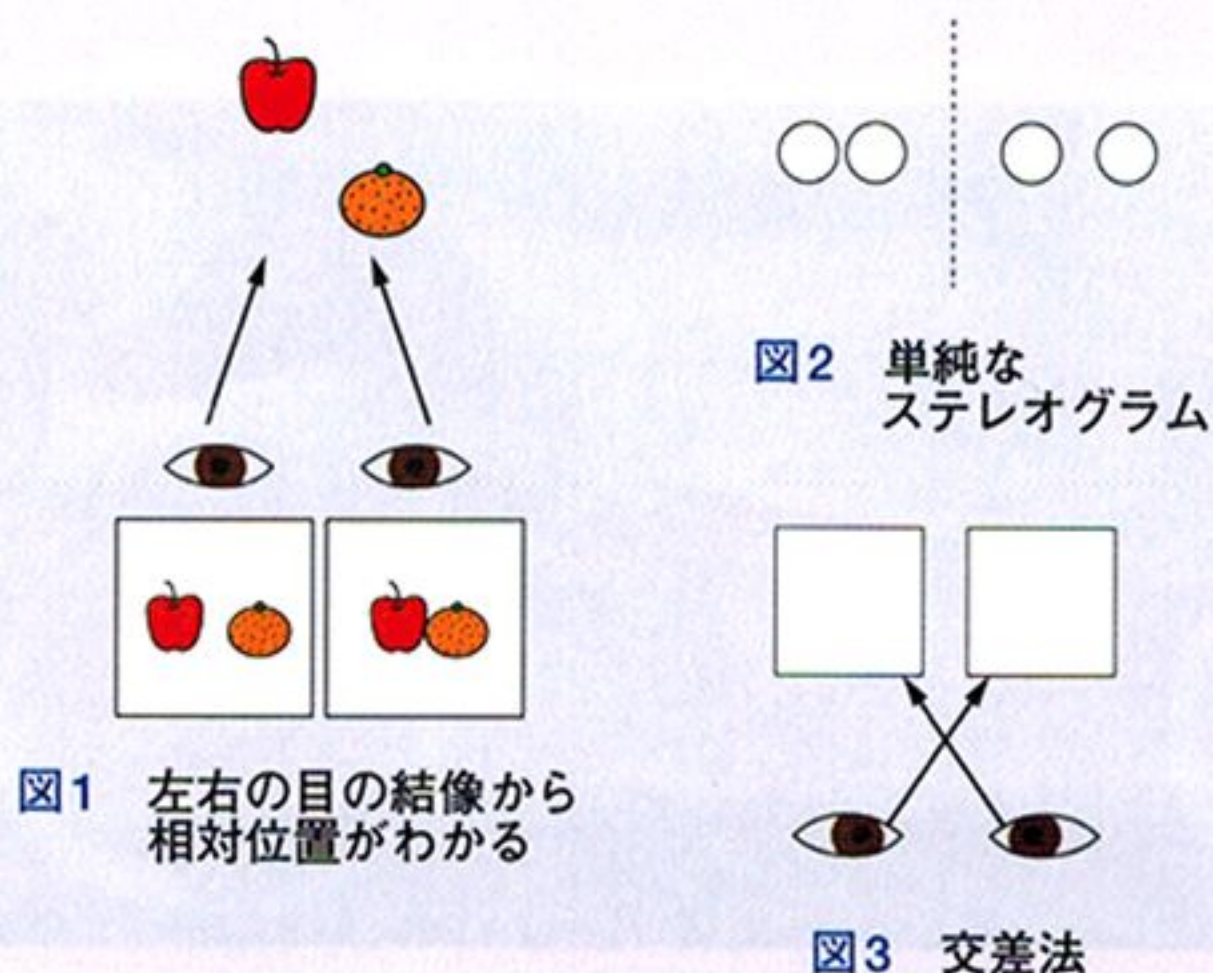
図1　左右の目の結像から相対位置がわかる

図2　単純なステレオグラム

図3　交差法

図4　平行法

図5　視点の設定

図6　ロンパリ法

に解像度が落ちる。

仮に1画面で2画面分の情報量を送らなければならないとしたときに，縦と横のどちらかを犠牲にするなら，横解像度のほうがよいのではないか？　というのが今回のデバイスの出発点になっている。

主流（？）である液晶シャッター方式の問題点を挙げてみよう。

1）ちらつく

家庭用テレビでは1/60秒ごとの切り替えとなり，画面のちらつきがはっきりとわかる。目の疲労も大きい。

2）メガネをかけないと機能しない

液晶シャッターをつけていない人が画面を見ると，非常に見づらい。さらに激しくちらつく画面は最近ではいろいろ問題がある。

3）フレームレートが低下する

フィールドごとに切り替えることはできるが，片目ずつでは1/30秒単位のフレームレートしか得られない。また，総合的に1/60秒単位で動かした場合には左右の目の情報が一致しない。右目と左目で1フィールドのレイテンシが発生する。

*

などなどだ。最初のだけではほぼ却下してもいいのだが，水平分割型では片目ごとのちらつきはなし，フレームレートも1/60秒でキープできる。さらにメガネをかけていない人が見てもなんとか状況は把握できる（画面が縦長になっているが）。さらにさらに，右目用画像左目用画像の順に配置すれば交差法で裸眼立体視することも不可能ではない。左右の目のレイテンシは半ラスタから1ラスタに抑えられる。

欠点はいうまでもなく，特殊なデバイス（ほぼヘッドマウントディスプレイ以外ないだろう）を要求することだが，ヘッドマウントディスプレイが非現実的ではなくなってきた現在であれば，ちょっとした付加装置だけでかなり理想的な3D再生環境を実現できるのではないかということにもなる。

はっきりいって，画質は悪い。

Voodoo3 3000のSVHS出力をコンポジットに変換し，アダプタに入力。ビデオ信号をラインメモリにキャプチャしつつ，2つのRGB信号に変換し，それをさらにコンポジットに変換して出力し，EYE TREKに与える。ちなみに，EYE TREKはそれをSVHS信号にして表示していると思われる。

パソコンの画像でもビデオの画像でも同様に処理できるというのはよいのだが，これではちょっと無駄な変換が多すぎる。もっとシンプルにできればいいのだが。

それでも，Direct3Dビュア（専用モードを加えたもの）を使って3Dオブジェクトを見ることができるのは素晴らしいことだ。

一昔前は夢物語だったリアルタイム3D画像生成がいまではまったく簡単にできてしまう。ただ，単にパースがついた絵を動かせば3Dなのではない。立体映像の持つ説得力は，名実ともに「次元が違う」ものなのだ。

*

さて，今回試作した3Dスプリッタはある意味で究極のデバイスになるはずだったのだが，パソコンに限っていえば，ビデオカード2枚使うのが現実的かもしれないなあと感じている。

Column　アスペクト比を変える

先ほど裸眼で疑似3D体験をする場合の話をした。しかし画面を2分割するのでは，結局縦長のせまっくるしい画面になってしまう（マルチモニタという話もあるが）。今回はEyeTrekで表示するために画面の左右を分離してそれぞれ1画面とするならば，必然的にそのままでは横に倍に引き伸ばされたつぶれた画像になってしまう。というわけで，アスペクト比を2：1にしろ，という上からのお達しであった。理屈としては，左右の画角を上下の画角よりも広く取ってやればよい。

3次元空間を2次元に投影する場合，まず射影変換を行う必要がある。たいていのレンダラは，Direct3Dも含めて，パースペクティブ投影である。これは，近くのオブジェクトは大きく，遠くは小さく見えるという，いまでは当たり前の遠近法の一種だ。

Direct3Dでは，図のようにカメラの画角が作る四角錐と，フロントクリップ，バッククリップで囲まれる「視錘台」の中にあるオブジェクトがレンダリングされる。では画角の調整はというと，直接画角を指定するのではなく，フロントクリップの大きさで指定できる（IMだともう少し複雑だが）。デフォルトはフロントクリップの一辺が1.0であるが，IDirect3DRMViewportインタフェイスの次のメソッドで指定できる。

```
SetField (D3DVALUE rvField) ;
```

この引数は，クリップ平面の端の数値である。つまりデフォルトは0.5。したがって，

```
lpD3DRMViewport->SetField (D3DVAL (1.0)) ;
```

としてやれば，縦横ともに倍の領域をカバーできる（画角は倍になるわけではない）。ちなみに，カメラからフロントクリップまでの距離のデフォルトは1.0だが，これもSetFront（）メソッドで指定できる。しかし，今やりたいことは縦と横のフロントクリップのサイズを個別に指定することだ。これには次のメソッドが利用できる。

```
SetPlane ( D3DVALUE rvLeft,D3DVALUE rvRight,D3DVALUE rvBottom,D3DVALUE rvTop) ;
```

見ればわかると思うが，これはフロントクリップ平面の隅の座標を直接指定できるのだ。ここで，

```
lpD3DRMViewport->SetPlane (-D3DVAL(1.0),D3DVAL (1.0) ,-D3DVAL (0.5) ,D3DVAL(0.5)) ;
```

とすることで，横方向だけを倍密に，すなわち描画時にはスリムに見える凹型のお化けミラーのようになる。

これでとりあえずアスペクト比は変えることができたが，いままでの説明で疑問に思ったことはないだろうか。なぜフロントクリップは（デフォルトで）正方形なのだろうか？　だってそうだろう。ディスプレイのアスペクト比は一般的に4:3，フロントクリップをそのままはめ込んだのでは，少しつぶれて変形しているはずだ。そうならない理由は，「Direct3D RMがそうしているから」という，なんとも二の句を次げない答えになる。

たとえば実際のカメラの場合，光が入ってくる領域は円形だが，その円形の中に納まるように配置されたフィルムなりCCDによって，さらにクリッピングが行われる結果になるだろう。それと同じことで，Direct3D RMでは，設定されたクリップの中に収まるように描画領域が設定される。つまり，描画領域が横長ならば横幅を合わせて上下をクリッピング，縦長ならば縦を合わせて左右が切り取られる。これがどういうことか？　640×480で描画するときは，横長であるので，横幅がフロントクリップのサイズに合わされて，画角調整される。つまり，上下はそれよりも狭い。それが320×480の半画面になると，今度は縦長になって，上下で画角調整が行われる。つまり，調整が行われた640：480=4：3分だけ密度が濃くなる，つまりオブジェクトが小さく表示されてしまう。したがって，全画面から左右分割画面に切り替える場合，フロントクリップのサイズを3/4にしなければならない。これは描画時のアスペクト比を2:1にする場合も同様で，フロントクリップが正方形でない場合も，まずは正方形に変換されてから処理されているようだ。以上を表にすると，フロントクリップのサイズはこんな感じだ。

モード	比	フロントクリップサイズ
全画面	1：1	1.0×1.0
全画面	2：	2.0×1.0
分割画面	1：1	0.75×0.75
分割画面	2：1	1.5×0.75

説明だけではなんなので，復刊号の付録CD-ROMに収録したオブジェクトビュアMeshViewをバージョンアップし，分割画面モードとアスペクト比の切り替え機能を付けておいた。F1キーで画面モード，F2でアスペクト比のトグル切り替えだ。ソースも付属しているので，参考にするといいだろう。別にEyeTreckがなくても，交差法でオブジェクトがぐるんぐるん回転する疑似3D体験を堪能できるぞ。

（菊地功）

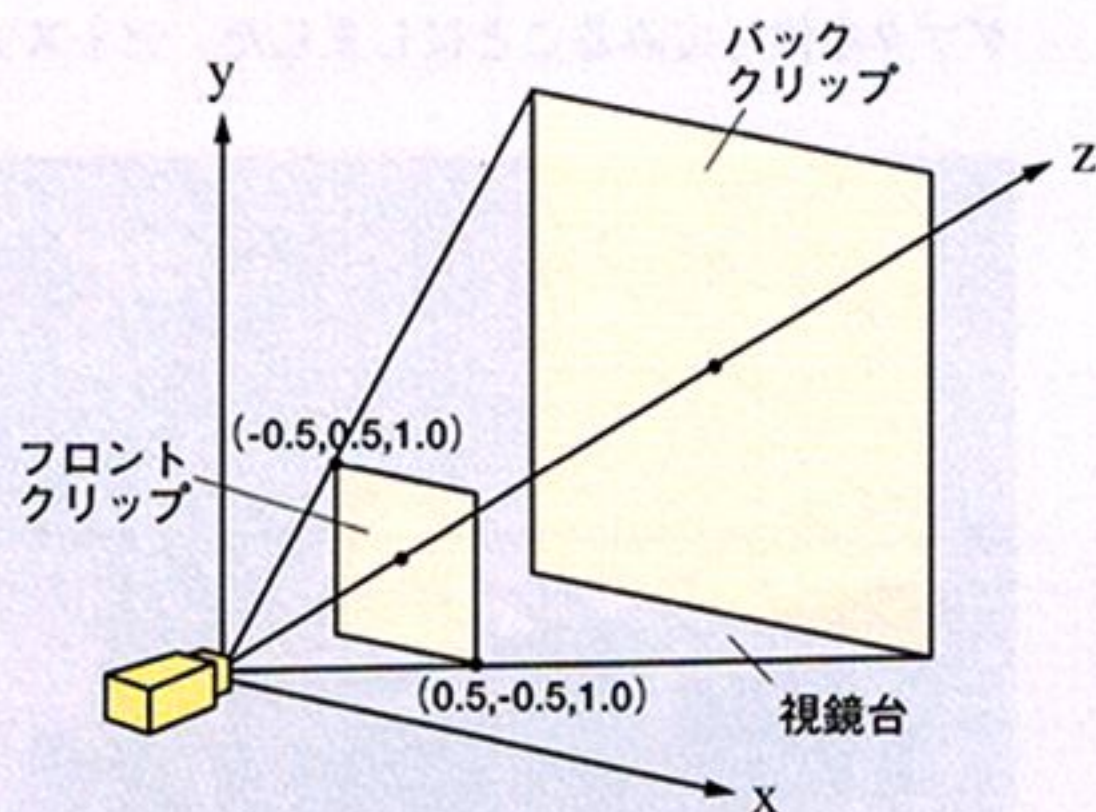



図7　射影変換の視錘台

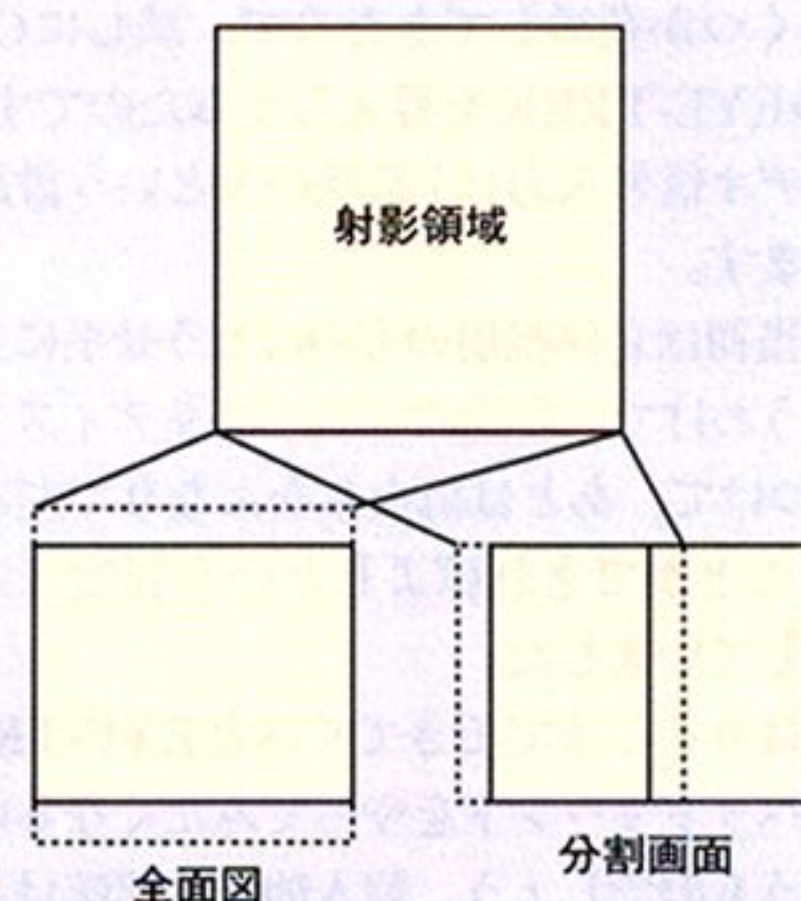



図8　射影領域と描画領域の関係

立体視アダプタを作ろう

桑野雅彦　Kuwano Masahiko

前章では，もっとも対応範囲の広い理想的な3Dアダプタはビデオ信号の左右分割型のものだという結論に達した。ここでは実際にそういった仕様のアダプタを製作してみよう。ヘッドマウントディスプレイに接続して効果を検証してみたい。

少しずらしたところから撮影した平面画像を2枚それぞれ左右の目に見せればものが立体的に見えるというのを利用して，仮想空間にいるような気分になれるという原理自体はずいぶんと昔から知られていて，版画や立体写真などというものも存在しましたが，応用範囲はごく限られたものでした。

ここにきて立体視が徐々にとはいえ広がりを見せてきているのはやはりコンピュータの演算能力の向上と，その低価格化にあるといえるでしょう。以前は漢字を表示するだけで精一杯だったのが，いまやリアルタイムで2枚分の画面の演算/描画処理を行うくらいは造作もないことになってしまいました。

また，3D再生方法として最高のものと思われるヘッドマウントディスプレイ側も，カラー液晶パネルが安価に手に入るようになったことで小型軽量化も一気に推し進められました。これらがあいまって，現在，業務用の分野を中心にいろいろと応用製品が出てきているようです。

これらのシステムを自宅でもなんとかしてみたいと思うのは当然のことですが，これがなかなか思うにまかせません。PCの性能は格段に上がっていますから，ソフトウェアは頑張ればなんとかなりそうですが，肝心の立体視用のヘッドマウントディスプレイがおいそれとは手に入らないのです。民生用ビデオ対応の製品はなんとか手の届く値段にまでなってきていますが，立体用となると途端に値段が跳ね上がるのです。

民生用ヘッドマウントディスプレイは確かに昨年来，いくつか登場してきたので，試しにOLYMPUSのEYE-TREKを導入してみたのですが，やはりビデオ信号入力は1系統のみという設計になっています。

今回も当初は立体視用のものはどうせ手に入らないというわけで，普通のカラー液晶ディスプレイを2つつけて，あとは鏡なりなんなりで左右の目で見ることができればよしという程度に考えて，製作していました。

が，やはりここまでできてくるとEYE-TREKのようなヘッドマウントをやってみたくなるのは人情というものでしょう。個人的にも興味はありましたし，（U）さんの熱い要望にも根負けして，挑戦してみることにしました。

PCで立体画像を作る

平面動画が2枚あれば立体視できるとなれば，当然やってみたくなるのが，PCによる立体画像の作成ですが，ここでいちばん面倒なのが映像信号を得る方法です。X68000のようにハードウェア的に初めから立体視アダプタがサポートされているような場合は簡単なのですが，一般には希なことです（編注：最近は立体端子付きのビデオカードも出てきました）。

また，2枚のビデオカードを使うというのもなかなか面倒なことでもあります。左右の画面が同期して動かないとおかしなことになりますので，できればひとつのCPUで2画面分の描画を行いたいところですが，ビデオカードを2枚用意できる方は限られているでしょう。複数のビデオカードの同時装着はMacintoshでは古くから行われていたことですが，Windowsマシンなどではサポートされ始めたのはごく最近ですし，立体視のためだけにビデオカードを複数実装する方はそうは多くないでしょう。PCの組み立てをやっている人ならビデオカードの2，3枚は持っているかもしれませんが，古いカードとの併用では3D性能に影響が出てきます。同等の性能のカード（できれば最新）2枚ということになると，ちょっと難しいでしょう。

また，ビデオなどに記録する場合も右と左を別々のVTRで記録するのは再生の同期の問題などがあり，不便です。

そこで，今回はひとつの画面の右半分と左半分をそれぞれ独立したビデオ信号として取り出すアダプタを作ってみることにしました。ディスプレイ側の多くがコンポジットビデオ信号（水平周波数15kHz）を前提としていると思われることから，今回のアダプタも水平周波数15kHzを基本として考え，入力信号はコンポジットビデオ信号ということにしてみました。

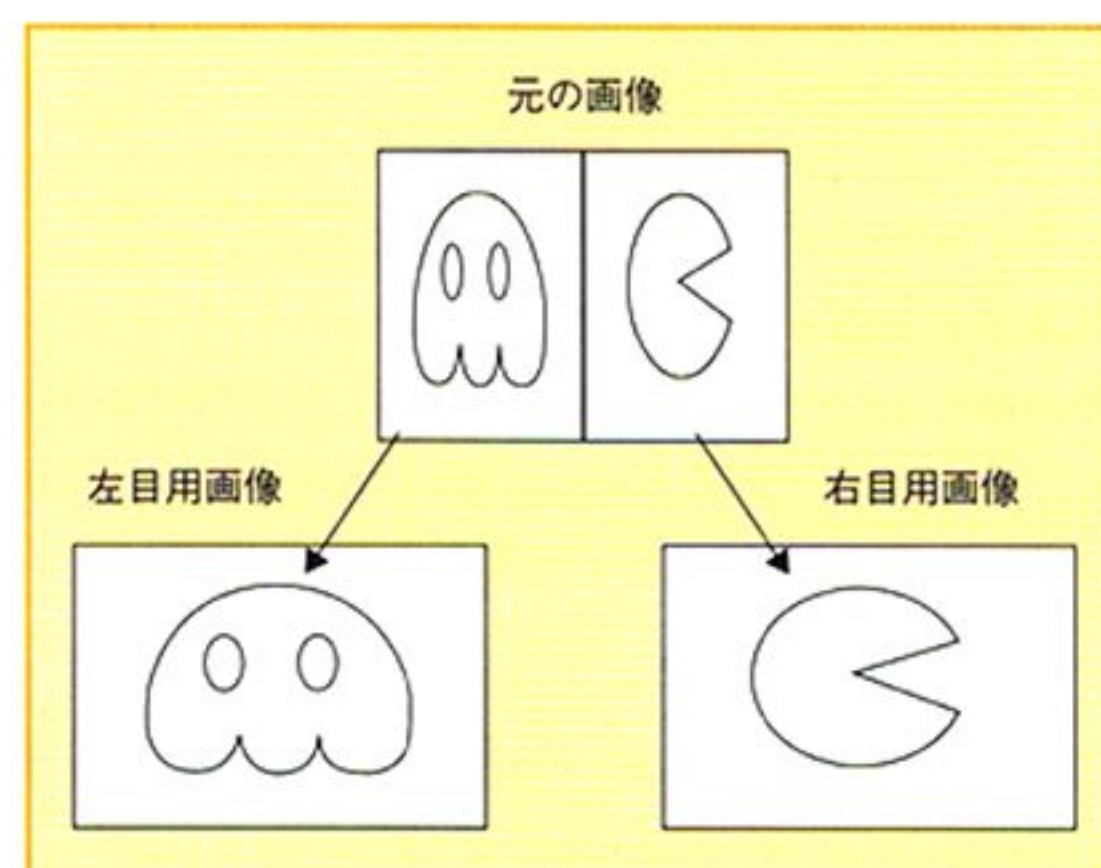



図1　立体視アダプタの基本動作

アダプタの基本的な動作を図にすると図1のようになります。元画像には右目用と左目用の画像が左右方向を半分につぶした形で表示されていて，アダプタではそれぞれを取り出して画面の左半分を引き伸ばした画像と，右半分を引き伸ばした画像を得ようというわけです。コンポジットビデオ信号を出力できるビデオカードはいくつもありますし，VTRなどを立体画像のソースにする場合もビデオデッキを2台用意する必要もなくなるので，なにかと便利でしょう。

アダプタの出力はアナログRGBやコンポジットビデオ信号ならたいていの機器で対応できるでしょう。今回はコンポジットビデオ出力をメインに考えてみました。

完成した3Dアダプタ。平行法で見よう

ディスプレイ
表示期間
水平帰線期間
垂直帰線期間

図2　CRT上の表示と帰線期間

画面
4.7μs
15.75KHz
映像信号
カラーバースト信号
水平同期信号

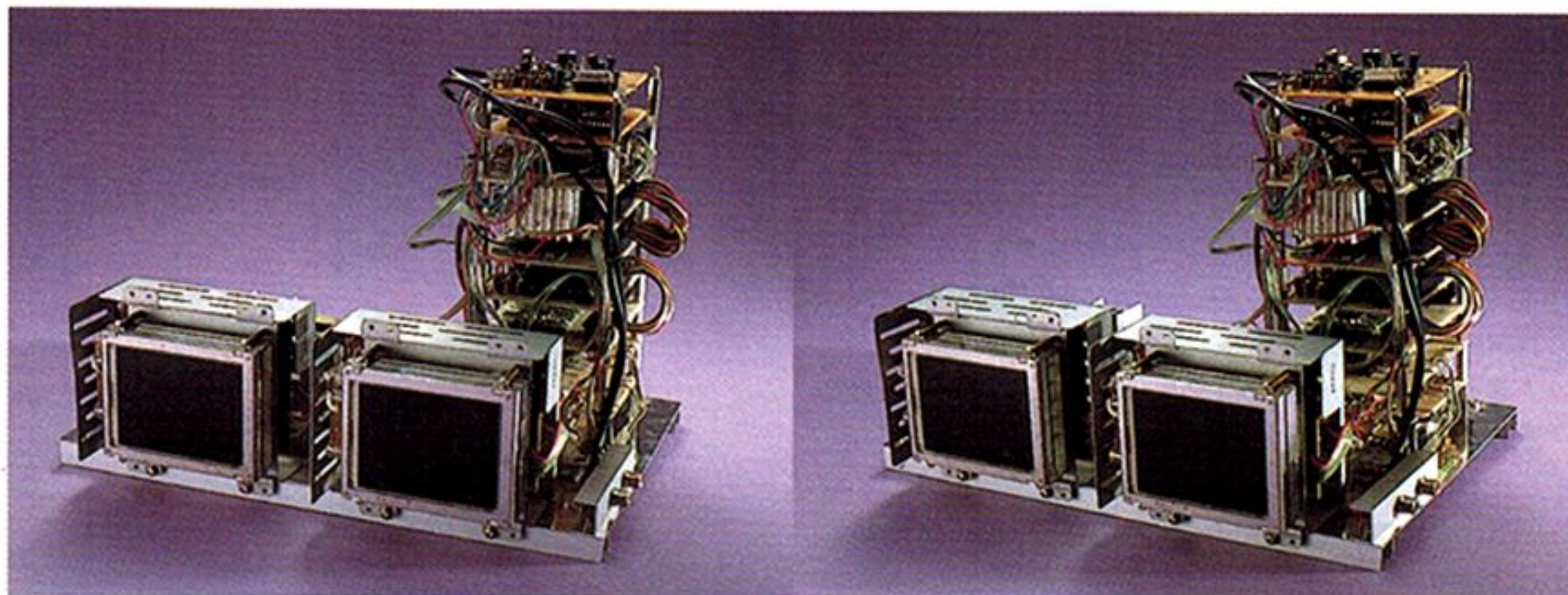
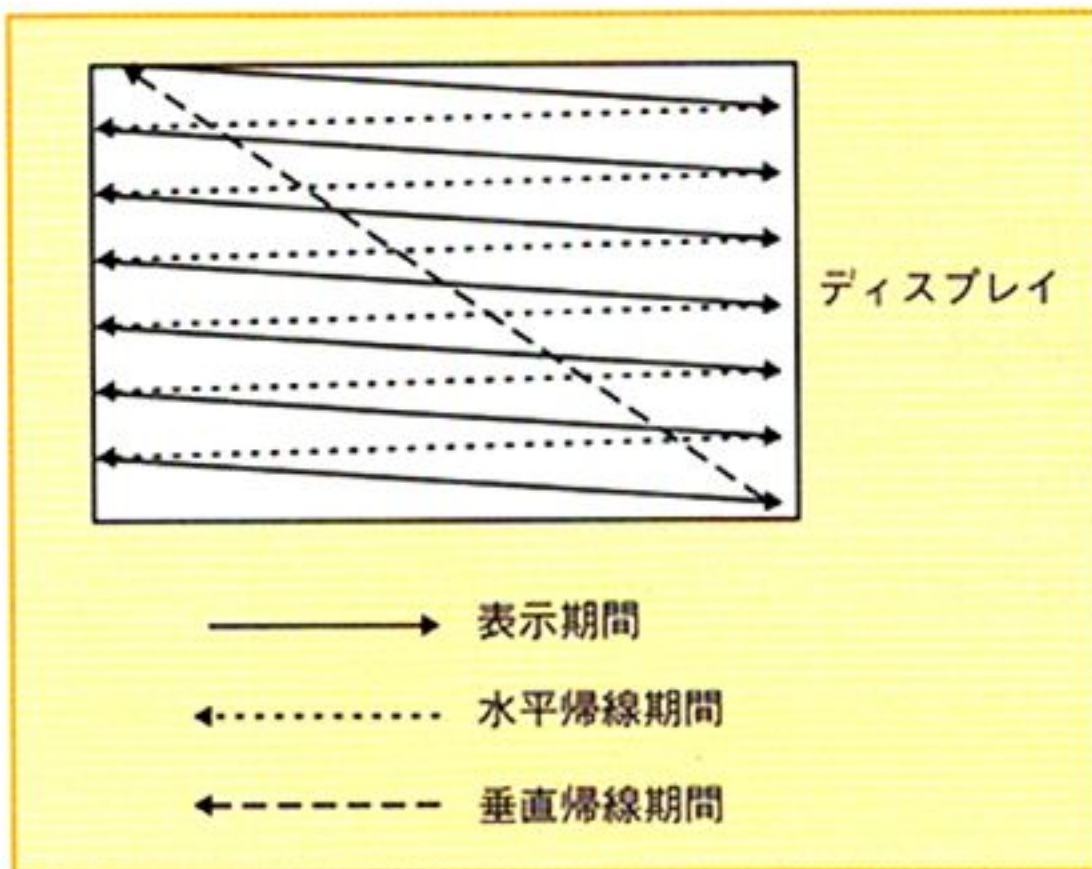



図3　コンポジットビデオ信号（映像）

コンポジットビデオ信号はどうなっている

まず，ビデオ画像を左右に分割するアダプタの設計に取りかかります。その前に，簡単にコンポジットビデオ信号について調べておきましょう。CRT上の画面表示は図2のように，左上から右の方に向かって1ライン分の表示が行われ，右端まで行くと表示をしない状態で左に戻り，先ほどよりも少し下のラインを再び表示し……と繰り返し，いちばん下まで行くと上に戻るという方法で行われます。この点に関してはテレビ画像でもパソコンのディスプレイでも同じです。

送信側と受信側で，この帰線期間のタイミングをあわせておけば送出した画像が受像器に表示されるいう理屈です。このタイミングをあわせることを「同期を取る」といいます。同期が取れていないと，画面が左右方向に歪んだり，上下にクルクルと回転したようになったりして，とても画像として見ることができない状態になってしまいます。

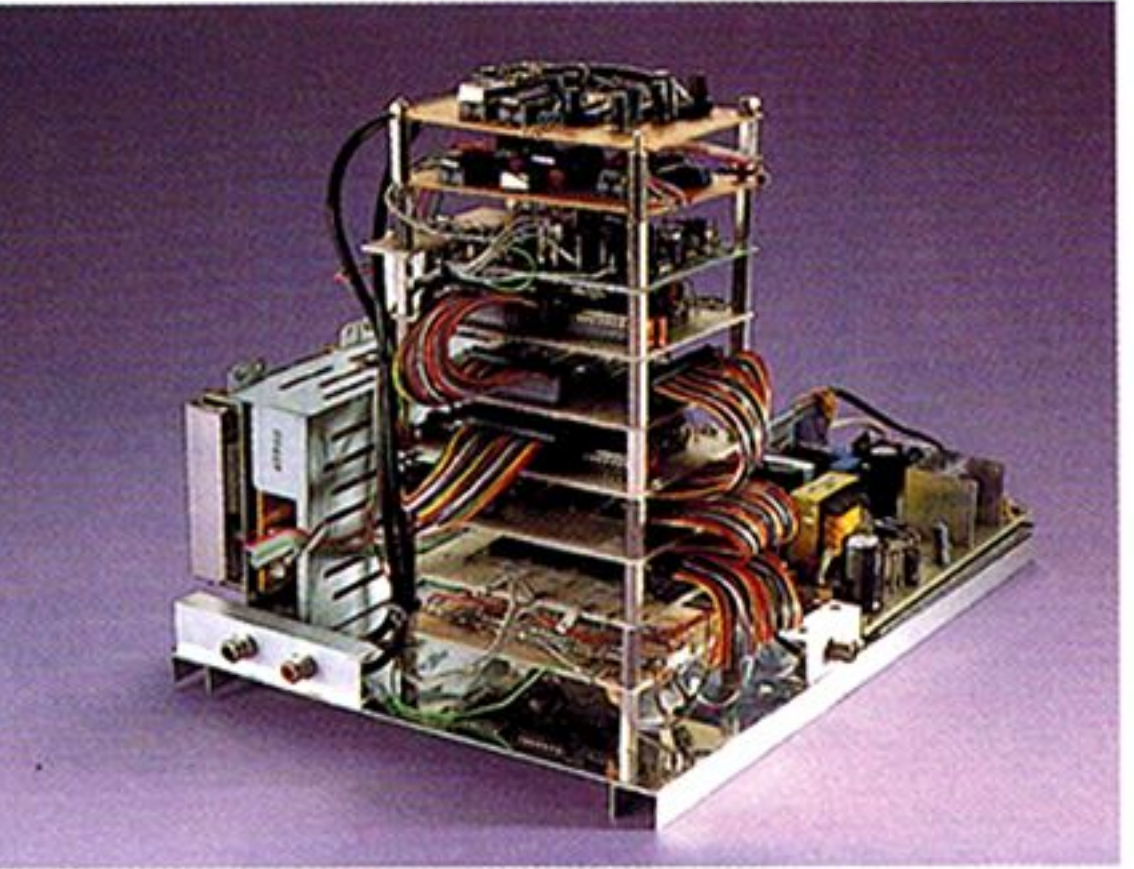
この同期を取るための信号を「同期信号」と呼びます。水平帰線期間を決めているのが水平同期信号，垂直帰線期間を決めているのが垂直同期信号というわけです。

PCのディスプレイのようなアナログRGB信号の場合には接続する相手がすぐ側にありますので，色は3原色（RGB）に分けてそれぞれ1本ずつあり，同期信号も水平，垂直それぞれ別々に用意されていますが，映像機器で使われるコンポジットビデオ信号ではひとつの信号線で輝度や色，同期信号などをまとめて送っています。これに加えて，ひとつの画面（フレーム）を2回（フィールド）に分けて送るなどなど，いろいろな細工もあるため，信号処理はPC用のアナログRGBに比べるとだいぶ面倒です。

テレビが真空管や個別トランジスタで作られていたときにはこの信号処理はアナログ回路の参考書のようなものが並んでいましたが，ICの発達によってすっかり様変わりし，いまはワンチップICの外側に少々の外付け部品だけで簡単に精度よく動くようになっています。

さて，図3がコンポジットビデオ信号の1ライン分の映像信号を簡単に示したものです。水平同期周波数は15.75kHzで，水平同期信号のパルス幅は4.7μsというのが規格値です。

この例では画面上では色の違う縦縞のストライプ模様になっていると仮定しています。水平同期信号が発生したあとに，カラーバースト信号がきます。このカラーバースト信号は次に送られてくる映像信号から色を抽出するための基準信号です。図では黒く塗りつぶしたようになっていますが，実際には3.579545MHzの周波数の正弦波です。

余談ですが，3.579545MHzを4倍すると14.31818MHzとなります。この周波数はなんとなく聞いたことがある方も多いのではないでしょうか。互換機のマザーボードやビデオカードなどで非常によく使われている周波数です。

カラーバーストが終わるといよいよ映像信号です。映像信号も黒く塗りつぶしたようになっていますが，これも3.579545MHzの信号で平均値が明るさ（輝度）を示し，3.579545MHzの信号の位相と，カラーバースト信号の位相の差が色を示します。

この位相差を取り出すため，カラーテレビでは内部に水晶発振器を持っていて，カラーバースト信号に自分の発振器の位相をあわせ，次に送られてくる映像信号との位相差検出に利用しています。

ちなみに，輝度信号のことはY信号，色信号のことはC（クロマ）信号と呼ぶ習わしになっています。コンポジットビデオ信号からYとCの信号を分離することは簡易的にはそれほど難しくないのですが，高画質を目指すと，いろいろと難しい問題が出てきます。

1ラインだけでなく，次のラインの信号も使って分離しようというのが2次元YC分離，さらに1フレーム分の映像信号を取り込んでおき，現在の映像データと1フレーム前のデータの比較も行って画質を高めようというのが「3次元YC分離」です。今回の回路ではこのような高級な方法は使わず，いちばん簡単なフィルタによる分離方法を採っています。

また，映像機器と表示器の間でわざわざY信号

Column　平面画像2枚式の立体視はなぜ疲れる

平面画像を2枚使う方法では目の焦点距離や左右の目の角度は固定という前提で計算されています。実のところ，おのおのの画面自体は所詮平面上の画像にすぎませんので，自ずと限界があるのです。

実際の人間の目では着目しているものに焦点を合わせようとします。近くを見ると寄り目になることからわかるように，距離によって目の角度も変えています。

また，話が複雑なのは単純に着目している物体の見え方だけではなくて，周囲の情報や自分自身の動きと目標物の動き方，経験から積み重ねた常識なども利用して距離感を得ていると思われることです。

日頃両目を使っている人が眼帯をすると，階段の上り下りや車の運転などに不自由を感じますが，片目が不自由になっても，しばらくすると階段を1段抜かしで駆け下りることもできるようになりますし，狭い場所での車庫入れも不自由なくできるようになるということは，距離測定が単なる左右の視差だけで行われているのではないということを表しているといえるでしょう。

私自身は専門家ではありませんが，平面表示を2つ使った場合に，これらの要素が抜けてしまっていることが，違和感や疲労感を覚えるといわれる原因のひとつではないかと思っています。

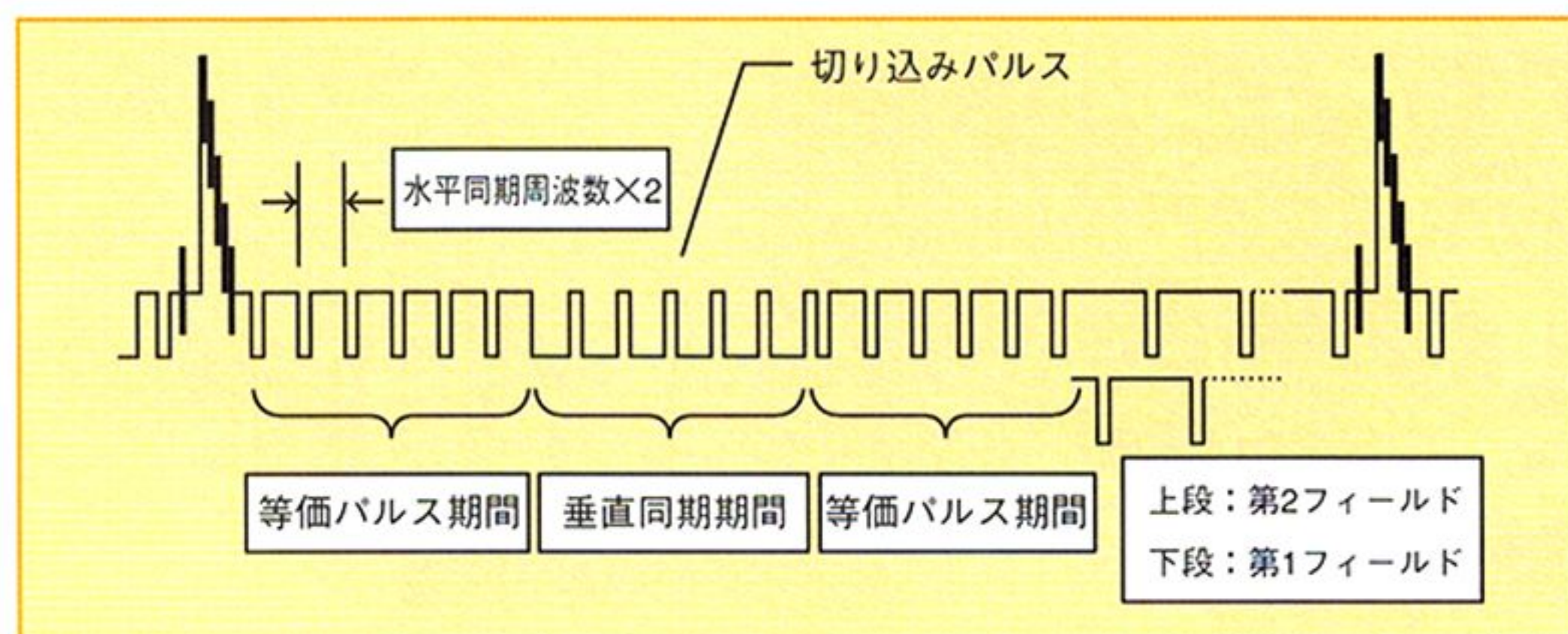



図4 垂直帰線期間の信号

図5 ラインメモリの信号

とC信号を混ぜたコンポジットビデオ信号でやりとりするのは画質の低下を招くだけだということでYとCを分離してやりとりしようというのがS端子です。

さて，この図では垂直同期信号は出てきませんが，では垂直同期のほうはどうなっているのでしょうか。これを簡単に図にしたのが図4です。

1フィールド分が終わった時点でまず等価パルスと呼ばれる，水平同期信号の2倍の周期のパルスが入ります。続いて，水平同期信号を反転させたような波形がきますが，これが垂直同期信号になります。さらに等価パルスが入ったあと，通常の幅の水平同期パルスが入り，映像信号が始まるという具合になっています。

垂直同期信号の中にあるパルスは切り込みパルスと呼ばれるもので，パルスがこないために水平同期回路が乱れるのを防ぐために入れてあります。

水平同期と垂直同期はこのようにパルス幅によって区別するようになっていますので，受像器側では簡単なフィルタを通すだけで垂直同期と水平同期を分離することが可能となるわけです。

ビデオスプリッタの設計

図11〜16が今回製作した映像信号を左右に切り分ける，ビデオスプリッタの各基板相互の接続関係，および各基板の回路図です。ラインメモリやD/Aコンバータ，RGBエンコーダはまったく同じ回路のものが左右それぞれに必要となります。

IC番号はCADソフトが勝手に割り振っただけなので，参考程度としてください。また，RGBデコーダ（ビデオ信号をRGBに変換する部分）はキットを使いましたので，回路図には載せていません。

製作したものは各ブロックごとにデバッグしながら進めていたので，すっかり高層ビル（というより，怪しい雑居ビル）という感じになってしまいました。

少々試行錯誤を繰り返しながらできていったものなので，制御回路やRGBエンコーダ関係はもっと整理できるはずですが，作り直す気力がありませんでした。これから製作されるなら，もうちょっと整理したうえで，アナログ回路の配線にも配慮したほうがよいでしょう。

ここで簡単に各回路の主要部分の説明をしておきます。

RGBコンバータ部

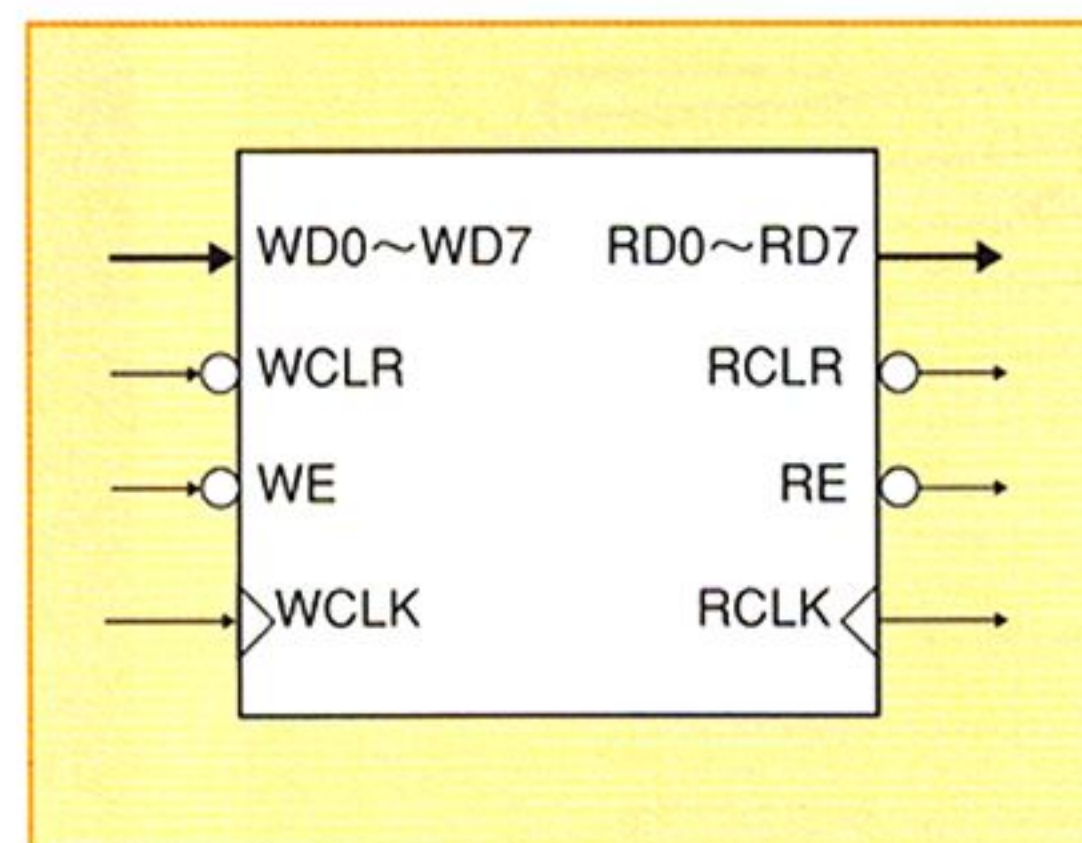
この手の回路ではコンポジットビデオ信号をそのまま記録/再生するようにしても悪くはないのかもしれませんが，製作記事や参考回路にあまり出てこないところをみると，オーソドックスにアナログRGBと同期信号に分離してから処理するのがよさそうです。

コンポジットビデオ信号からアナログRGB信号，同期信号を取り出すことはテレビ受像器などでごく普通に行われていることですので，いろいろなICが作られています。ICのデータシートには参考回路がついているのですが，どれを見ても昔のカラーテレビの回路などを見慣れた目にはずいぶんと簡単になったものだとため息が出るほどです。

ただ，それでも一応はアナログ信号処理ですからICの外部に細々とした部品が必要となります。手慣れた人にはたいした手間ではないでしょうし，パーツケースを引っくり返せばたいていのものは揃うと思いますが，初めての方にはちょっと面倒ですので，今回は秋月電子通商のアナログRGBコンバータキットを使用しました。部品一式と専用基板がついていますので，説明書に従って部品を実装してハンダづけするだけで完成です。

このキットの出力段には6dB（電圧利得2倍）のビデオアンプがついており，A/Dコンバータのような高インピーダンス入力のICで受けると出力電圧はほぼ2Vp-p（振幅2V）程度になります。本来は75Ωの負荷を持ったCRTなどをつないだときにちょうど1Vp-p程度となるように考慮したものなのですが，この2Vp-pという電圧がA/Dコンバータにはちょうどよいレベルであるため，そのまま電圧増幅器として利用することにしました。

A/Dコンバータ

A/Dコンバータ（アナログ/デジタルコンバータ）は入力されたアナログ電圧に応じたデジタルデータを出力するものです。いわゆる計測関係などでは12ビット以上の分解能を持ったものが使われるのが普通ですが，ビデオ信号ではそれほど高い分解能は必要としないため，高速の6〜10ビットのA/Dコンバータが使用されます。今回はソニーの8ビットのA/DコンバータIC，CXD1175APを使用しました。このA/Dコンバータの内部はパイプラインになっていて，クロックに同期してデータ処理されていきます。取り込まれたアナログ電圧が8ビットデータに変換されるまでの時間は約2.5クロックとなっています。

また，このA/Dコンバータは0レベル電圧とFFh（255）の電圧レベルを外部端子で設定できるようになっているだけでなく，基準電圧の発生回路も内蔵しているなど，ビデオ用を意識した作りになっています。内蔵の基準電圧は0レベルが0.6V，FFhレベルが2.6Vで，振幅にしてちょうど2V程度となります。試しにRGBコンバータの出力を直結して波形をシンクロで見てみると，ほぼこの幅で収まるようですので，内蔵の基準電圧を使って直結することで間に合わせることにしました。

このおかげで外付けのレベル変換や基準電圧発生回路がごっそり不要になり，回路はほとんどブロック図と大差ないものになってしまいました。

A/Dコンバータに与えるクロックはラインメモリの書き込み用のクロックと同じものを使います。

ラインメモリ

ビデオ関係の回路だけでなく，スキャンコンバータやファクシミリのように1ライン単位でのデータ処理を行う回路の要となるのが，ラインメモリと呼ばれるメモリICです。

通常のメモリICはアドレスピンとデータピンが1組あって，あるアドレスを指定してデータの読み書きをするという動作をするわけですが，今回のアダプタのようにデータのリードとライトが同時に発生するような場合にはリード/ライトを交互に切り替えていかなくてはならず，かなり面倒なものとなります。

また，ライン単位で処理するような場合の多くはアドレスは1ずつ増えるだけで，任意のアドレスを読み書きする必要はほとんどありません。このような使用目的に合わせて作られたのがラインメモリです。ラインメモリはFIFO（First-In First-Out）メモリの一種で入力側から書き込まれたデータがその順番どおり出力側から取り出さ

図6　映像信号の分割

図8　制御入力付き発振回路

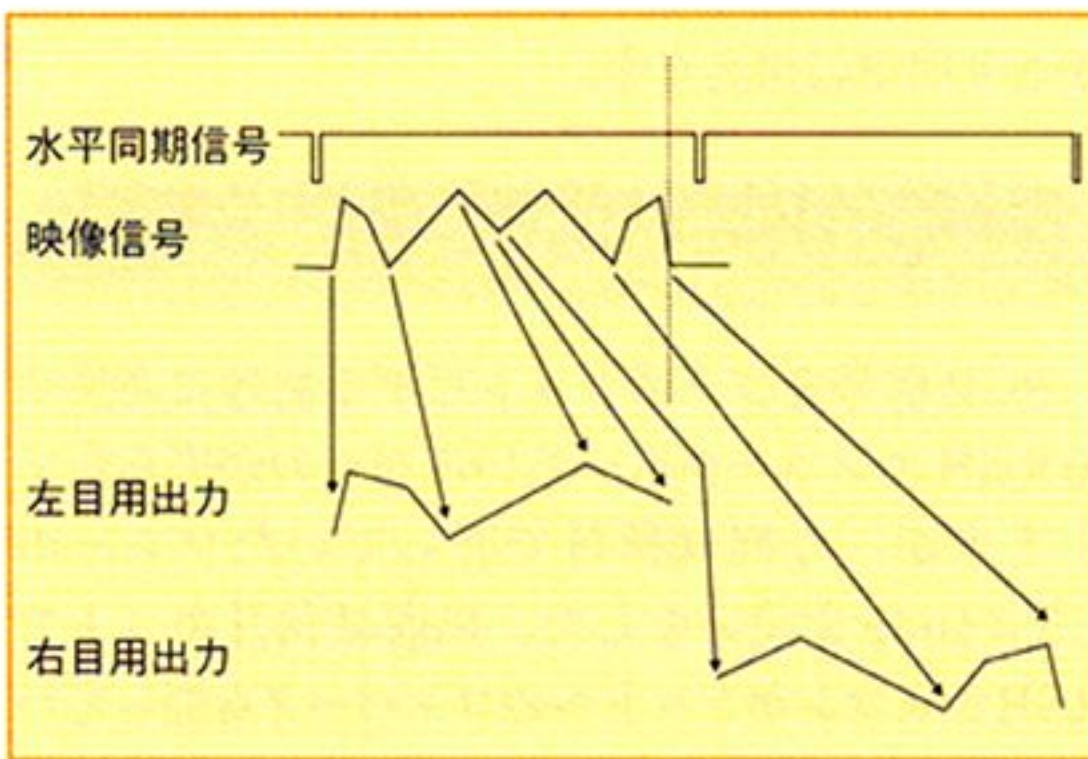
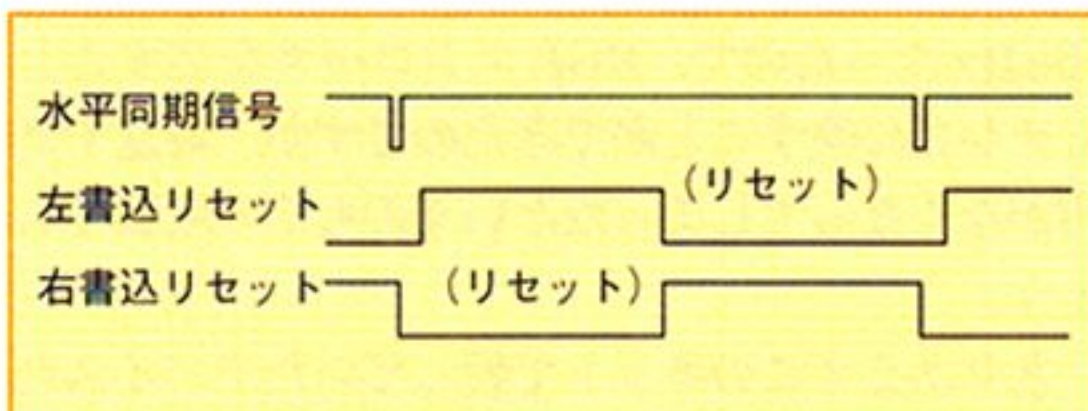
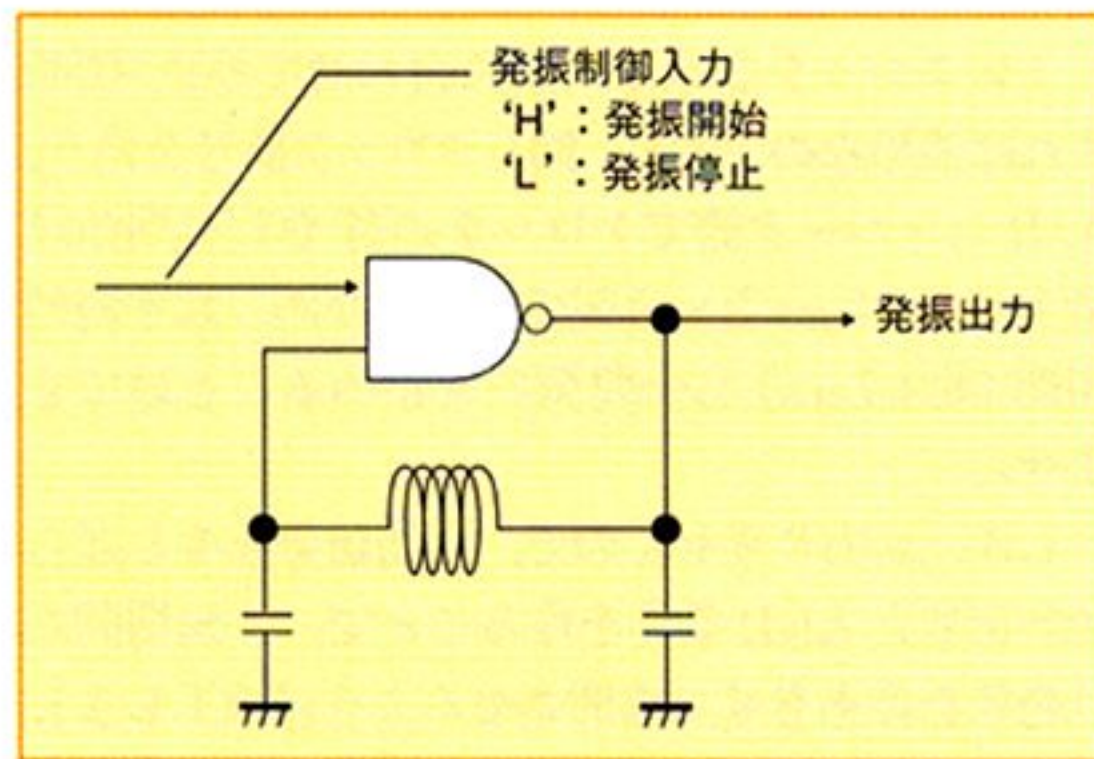



図7　書込リセットタイミング

れるものです。一般にFIFOメモリと呼んだ場合には非同期型のものが多いのですが，ラインメモリの場合には同期型（クロックを基準として動作するもの）が普通です。

ラインメモリにはいろいろな種類がありますが，今回使用したものはNECのμPD485505というものです。この手の回路に使われるラインメモリとしては同じNECのμPD42101が定番だったのですが，あちこち手を回してもなかなか手に入りません。

世の中の映像機器が1画面分のメモリを必要とする方向に向かってしまったためか，国内では製造中止扱いになってしまったということです。代替になるものがないかNECのホームページで探したら出てきたのがμPD485505で，こちらは特に問題なく入手できました。μPD485505は42101よりも容量がずっと大きく（42101が910バイトなのに対して，485505は5Kバイト），想定している製品が違うのかもしれませんが，使い方が変わるわけでもありませんし仕様上も特に問題はありません。

さて，ラインメモリの制御信号はどの製品もほとんど同じで図5のようになっています。図の左半分が書き込み用，右半分が読み出し用の制御信号です。両者は完全に独立していますので，使い方は非常に簡単です。

RD，WDがそれぞれデータ出力/入力用の端子です。データが独立していますので，外部でタイミングを見てデータバスを切り替える必要もありません。

RCLK，WCLKにはそれぞれ読み書きタイミング用のクロックを与えます。

RCLR，WCLRはそれぞれ読み書きのアドレスを初期化（0にする）するためのものです

RE，WEが読み書きのイネーブル信号です。これが'L'レベルになっているとクロックにあわせてデータが順次出力されてきます。アドレスは読み出したときに勝手にインクリメントされますので，特になにもしなくても次のアドレスのデータが出力されるというわけです。

制御回路

今回の製作上いちばん苦労させられたのがこの制御回路です。制御回路はクロックの生成とラインメモリの制御を受け持ちます。当初はもう少し簡単だったのですが，細々した問題に対応しているうちにこのような形になってしまいました。これから作られるのでしたら，74AS174をもう少し上手に使うなどして整理したほうがよいかもしれません。

さて，この回路ですが，左目用は比較的簡単です。水平同期信号にあわせて画像を取り込み始めてそれを1/2の速度で取り出すことで，画面の半分までのデータが横一杯に表示されることになります。

右目用はちょっと考えなくてはなりません。水平同期信号が入った時点ではそのラインの右半分用のデータは当然また到達していませんから，表示のしようがありません。

そこで，右目用は1ライン前のデータを表示することにして解決します。ひとつ前のラインの右半分のデータはすでに到着し終えていてますからこれを左目用データと同様に半分の速度で取り出していけばよいだろうというわけです。右目用と左目用で1ライン分ずれることになりますが，立体視用として使う分には問題とはならないでしょう。

この動作を映像信号を基準に考えると，図6のようになります。1ラインの半分までのデータはそのまま横に伸ばして表示されて，残り半分はバッファに入ったあとで次の水平同期信号から表示されていくわけです。

画面の真ん中にきたときには，右目用のバッファの前半1/2までのデータは表示を終えたあとですので，空きになっています。ここに次のライン用のデータを詰めていくことになり，右端では表示し終えた直後に新しいデータが書き込まれることになるわけです。

ラインメモリタイミング生成回路

次に重要なのがラインメモリのタイミング制御を行っている部分です。今回は読み出しは書き込みの1/2の周波数に固定してます。RCLR信号は基本的に水平同期でリセットすればよいであろうことは容易に想像できます。考えなくてはいけないのは書き込み方向のリセットタイミングです。これは図7のようになっています。実は今回使用したラインメモリは容量が十分あるので，左半分用はリードと同じように単純に水平同期でリセットしてしまってもかまわないとも思えるのですが，

Column　R-2R型D/Aコンバータ

R-2R型（ラダー型とも呼ばれます）のD/Aコンバータはある抵抗値Rの抵抗とその2倍の値の抵抗（2R）の2種類の抵抗を梯子のように組み合わせることで出力にデジタルデータに対応したアナログ電圧が得られるというものです。基本構造は図9のいちばん左の図のようになります。この例はデジタル入力が4つ（4ビット）ですが，必要に応じて同じ調子で伸ばしていけばビット数を増やすことができます。

この入力条件を全部ひっくるめて考えると頭が痛くなるのですが，ここで「重ねの理」と呼ばれるものがあって，各電源を短絡して個別に計算した結果を足しても同じになるということがわかっています。これを使って個別計算してやるとD/Aコンバータとしてビット重み付けされた結果が得られることがわかります。興味のある方は実際の計算や任意のビット数になったときへの展開などを考えてみるとよいでしょう。

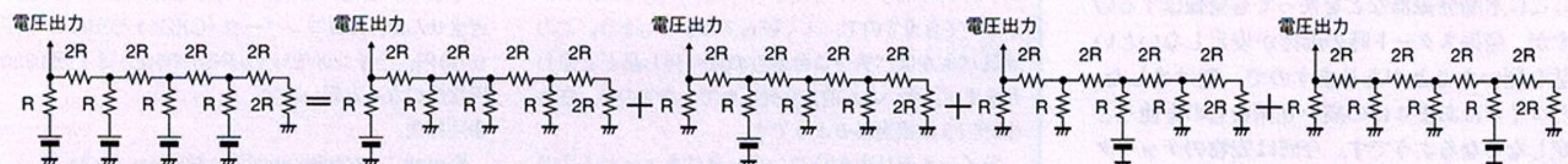



図9　R-2Rラダー型DAコンバータ

実は同期信号から実際の表示が始まるまでの期間も2倍に伸びてしまうため画面の左端に無表示領域ができてしまうのです。これを読み出し時に細工するという手もあるのですが，今回は単純に右用と同じように不要領域ではリセットするように考えてみました。制御回路を見るとわかるとおり，この波形はワンショットで簡単に作っています。

クロック発生回路

制御回路のクロック発振回路を見ていきます。今回のようなラインメモリを使ったアダプタの場合に考慮しなくてはならないことのひとつに，クロックの位相の問題があります。A/Dコンバータによる変換も，ラインメモリへの読み書きもすべてクロックにあわせて動くわけですから，クロックとビデオ信号が同期していないと，同じ画面の位置であっても，最大1クロック分位置がずれる可能性があるわけです。つまり，静止画を表示しているのに左右に微妙に揺れているような状態になる可能性があるわけです。今回使用したような小型の液晶ディスプレイでは1クロック分くらいの揺らぎはほとんどわからないとは思いますが，あまり格好よくありませんので，水平同期信号とクロックの同期を取ることを考えます。

このような方法はいくつか考えられます。もっとも安定性のよい方法は，PLL（Phase Locked Loop）によって水平同期信号の整数倍の周波数を得る方法です。テレビ放送のように信号タイミングが厳密に守られていて，安定しているときにはこの方法がいちばん望ましいといえるでしょう。ただ，この方法ではゲーム機やVTRの再生出力のようにタイミングが多少いい加減なものや，変動のある信号が与えられた場合にはPLLがロックできなかったり，動作が不安定になることがあります。

今回はもっと簡単な方法として，単純に水平同期信号にあわせて発振器を毎回リスタートさせるように考えてみました。水平同期信号にあわせていったん止めてタイミングをあわせてスタートさせるというわけです。

発振周波数は一切フィードバックをかけたりしませんので，温度などによる変化はあるのですが，今回のような用途では長い周期であれば周波数が多少変化しても関係ありません。

このような制御入力付きの発振回路はいろいろなものが考えられますが，今回は図8のようなコイルとコンデンサを使ったものを使っています。電子回路の発振回路のところで必ず出てくるコルピッツ発振回路と呼ばれるものの応用で，発振周波数は2つのコンデンサの直列接続による容量とコイルのインダクタンスで決まります。

ここに水晶発振器などを使っても発振はするのですが，発振スタート時の動作が安定しないという話を聞いたことがありますので，避けました。またコイルはあまりQの高い立派なものを使うと安定しなくなるようです。今回は安物のチョークコイルを使用しましたが，まずまず安定して動いているようです。

このクロックを水平同期に合わせて停止/開始すれば簡単なのですが，同じクロック信号を使う，A/Dコンバータ側でクロックの'H'や'L'の期間は最大でも1.1μsという規定があるため，水平同期期間（約4.7μs）もの間発振を止めることはできません。

しかたがありませんので，同期信号を少し遅らせた信号とAND条件を作ることで，一定期間だけ発振を停止させて再開させるように細工しました。このディレイ時間はさほど精度もいらないため，74AC14を使ったCRによるディレイで済ませました。

発振の停止のしかたはちょっと気をつけなくてはなりません。クロックの停止タイミングがクロックと同期していないと，停止直前のクロック波形がヒゲのようになる可能性があります。ICによってはこのようなクロック信号が入力されると誤動作したり，出力が不安定になったり，同期リセットがうまく働かなくなるなどの異常現象となる場合があります。このため今回は74AC174を使って停止タイミングをクロック信号で同期させておきました。

D/Aコンバータ

D/Aコンバータは専用のICもありますし，秋月などでも比較的簡単に入手できるのですが，今回は普通の集合抵抗を使ったR-2R型と呼ばれるD/Aコンバータを作ってみました。この回路はかなり有名なので私も形だけは知っていたのですが，改めて動作を考えていくとなかなか理解できず，苦労させられました（※2）。

部品は集合抵抗を使うと簡単です。今回のような用途では精度はそれほど要求されませんので，手に入らなければ個別抵抗でも構いません。

出力はそのままではレベルが大きすぎるので抵抗で簡単に分圧して約1Vp-p程度のレベルにして出力しています。当初この部分を半固定抵抗にしていたのですが，調整箇所が多くなりすぎる（両目で6カ所）ので固定してしまいました。

この出力はそのまま液晶ディスプレイのほうに接続していますが，将来アナログRGBディスプレイなどにつなぐことも考慮して，回路にはビデオバッファアンプをつけてあります。

バッファアンプは，RGBコンバータキットにも使われているものです。利得が固定（6dB）な分，かなり簡単に扱えます。

RGBエンコーダ

RGB信号をコンポジットビデオ信号に変換するRGBエンコーダもいろいろなものが売られていますが，今回は秋月で売っていたソニーのCXA1645を使いました。以前は秋月キットでRGBからコンポジットへのコンバータも売っていたのですが，最近中止になってしまったようです。

昔の8ビットパソコンや16ビットの当初の頃の縦200ラインの画面モードの時は水平周波数が15kHzだったので，RGBエンコーダを通すことでテレビに映すことができたのですが，最近では用がなくなってしまったというのもその原因でしょう。

とりあえずこのキットで使っていたデバイスだけはまだ入手できるようですので，今回はこのICを使って基板は自分で作成することにしました。

回路自体は，ICに添付の応用回路例そのものです。周辺部品は20KΩの金属皮膜抵抗（誤差1%）というものがちょっと特殊といえば特殊な程度で，ほかに特別なものはありません。そこらの部品屋さんで簡単に調達できるものばかりでしょう。

74HC221と74HC07で組んだ怪しげな回路は，映像信号がない区間の映像信号を黒レベルに固定するためのものです。

当初，ICの応用回路どおりに組んだのですが，画像の横縞がひどく，とても見られたものではありませんでした。RGBエンコーダを組む前にもRGB信号で液晶ディスプレイを駆動してチェックしていたのですが，特別変なところはなかったので，CXA1645をつけるだけだと思っていたのですが，やってみるとまったくだめなのです。

さんざん悩んであれこれいじった結果わかったのは，CXA1645は同期期間やそのあとのカラーバーストの期間（ブランキング期間）にRGB信号が黒レベルになっていないと，誤動作するらしいということでした。

カラー液晶のほうは，同期期間は同期，映像は映像としか見ていないので問題ないのですが，CXA1645のほうはこの期間が黒レベルであることを前提にして回路を簡略化したというあたりが原因なのでしょう。

このため，回路図のような回路で，ブランキン

Column 部品入手について

RGBコンバータとカラー液晶ユニットは秋葉原の秋月電子通商で入手できます。液晶パネルは3.3インチのTFTのタイプを使いました。パネルに説明書がついてきますので，よく読んでおきましょう。この液晶パネルはパチンコ台からの取り外し品だと思われますが，ずいぶん前から売られていますので，在庫（補充？）は相当あるようです。

ラインメモリやA/Dコンバータはちょっと入手が難しいかもしれません。これらを含め，入手が難しいものがあった場合は私のほうで手配することも可能です。お問い合わせは電子メールで下記までお願いします。価格は値動きのあるものなので，断定できませんが，A/Dコンバータ（CXD1175AP）が1個1500円，ラインメモリ（μPD485505）は1個1000円程度になると思います。

●連絡先
E-mail：pastelmagic@po.teleway.ne.jp

グ期間（約10 μs）の間，映像信号を74HC07で強制的に0Vまで引いてしまったというわけです。

アナログRGBを受けるのに，ビデオアンプ（NJM2267）を入れてみましたが，とりあえずなくても絵は出ます。私が作ったものでは片チャンネルだけ入れてみました。

ビデオアンプはゲインが6dBありますので，電圧が2倍になります。1Vp-pの入力だと出力が2Vp-pになるので，抵抗で分割して落としています。明るすぎたり，暗すぎるときはこの抵抗を加減してください。半固定抵抗にしてもよいのでしょうが，調整が面倒なのでやめました。

ビデオスプリッタを組み立てる

製作は回路図を見ていってください。回路図には現れませんが，各ICの電源とグランドピンの間には0.1 μF程度のセラミックコンデンサを入れておきます。A/Dコンバータの電源まわりだけ注意すればあとは多少いい加減でもなんとか動くとは思います。

私は機能別に実験しては相互接続という手順でやっていたので，ユニバーサル基板が6枚，秋月のRGBコンバータキットとあわせてなんと7階建てというものになってしまいました。基板間の接続はフラットケーブルを使ったのですが，さすがにこれでは電源が弱くてしかたありません。画面上にチラチラとスパイク状のノイズがたくさん乗ってしまい，とても見られたものではありませんでした。

やむなく固定用のネジをグランド用に流用するなどしてグランドの強化を図って対策したのですが，アナログ的な目で見れば，とてもほめられたものではありません。これから作られるのでしたら，大きいユニバーサル基板にまとめて，グランドは銅箔を使うなど，もう少し電源の取り回しも考慮して製作したほうが賢明だと思います。

消費電流は液晶ユニット2個もあわせて，全部で0.8A程度でした。とりあえず1A取れる電源ならば大丈夫でしょう。液晶ユニットのほうが9Vを要求するので，+12Vの電源から3端子レギュレータで9Vに落として与えています。デジタル回路で必要な+5Vはこの9Vからさらに3端子レギュレータで作りました。使用した電源に+5V出力もあったのでそちらから取ってもよかったのですが，実験中に着脱が面倒になってきたので，+12Vの単一入力に変更してしまいました。9Vのレギュレータに0.8Aも流れるため，かなり熱くなります。余ったアルミ材で簡単なヒートシンクを作りましたが，この程度の大きさでは長時間使うのは無理なようです。これから作るのであれば9V出力の電源を見つけてきたほうが簡単でよいでしょう。

A/DコンバータはICソケットを使いたいところなのですが，ICの幅があまり一般的なものでない400mil（10.16mm）というもので，普通に市販されているICソケットであうものがありません。しかたがないので，14ピンのソケット2つ並べてその間に実装するという，アマチュア的な方法で対処しました。シングルインライン型のソケットなどが手に入るのであれば，もう少しスマートにまとまるでしょう。

ラインメモリはフラットパッケージなので，変換基板としてサンハヤトのICB-010を使用しました。ところが，これまたICの足の幅が大きく，変換基板のパッドにうまく載ってくれません。こちらもアマチュア的にICの足を内側に折り曲げて対処しました。折り曲げるのは1本ずつ行わずに，

1

4

7

2

5

8

3

6

9

図11

図12 Video Signal Input

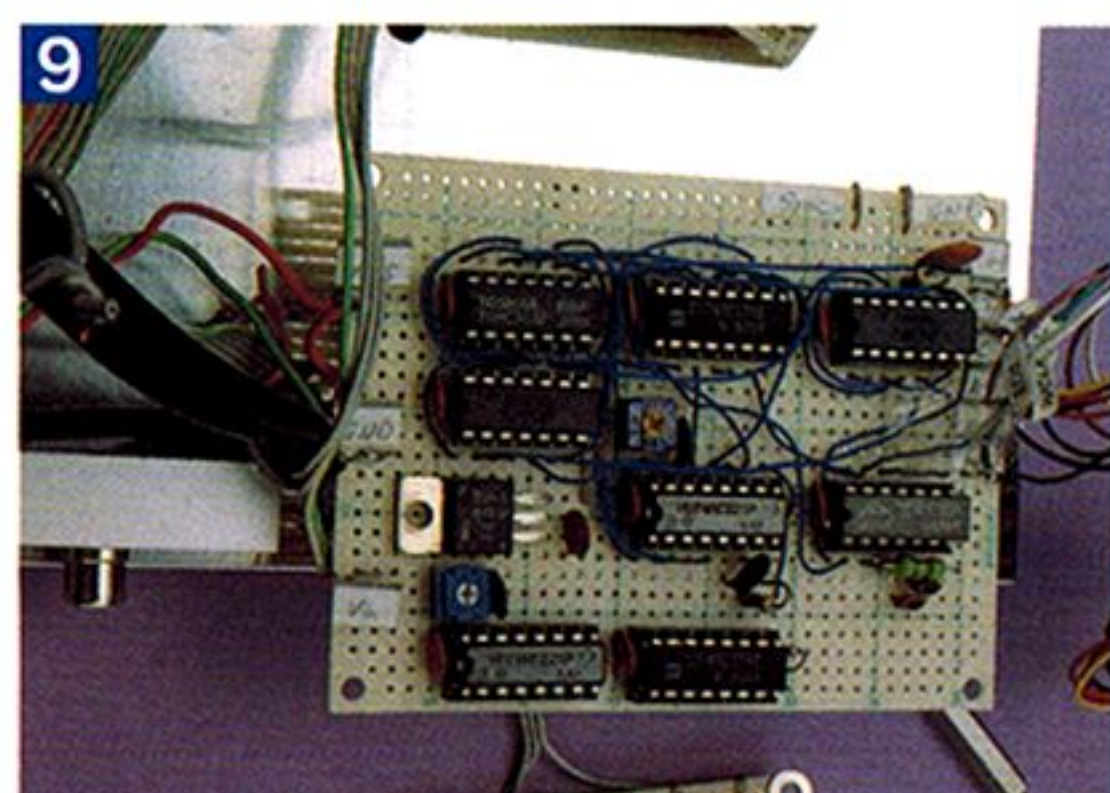
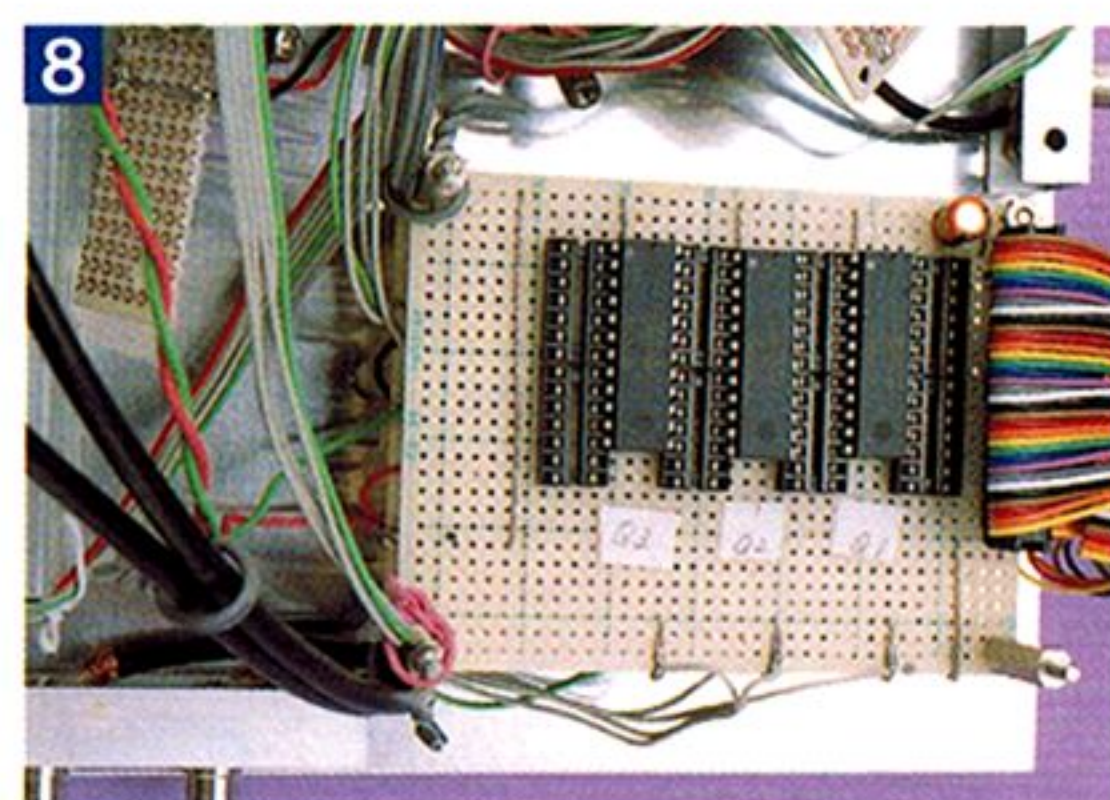
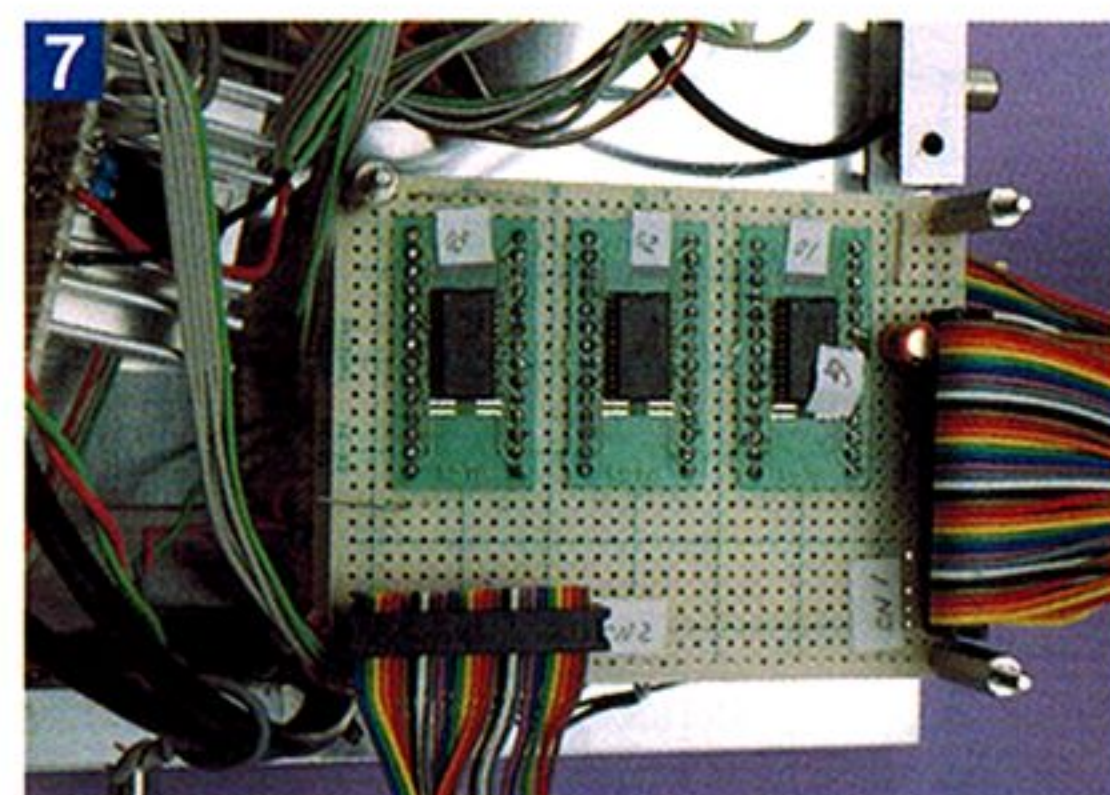
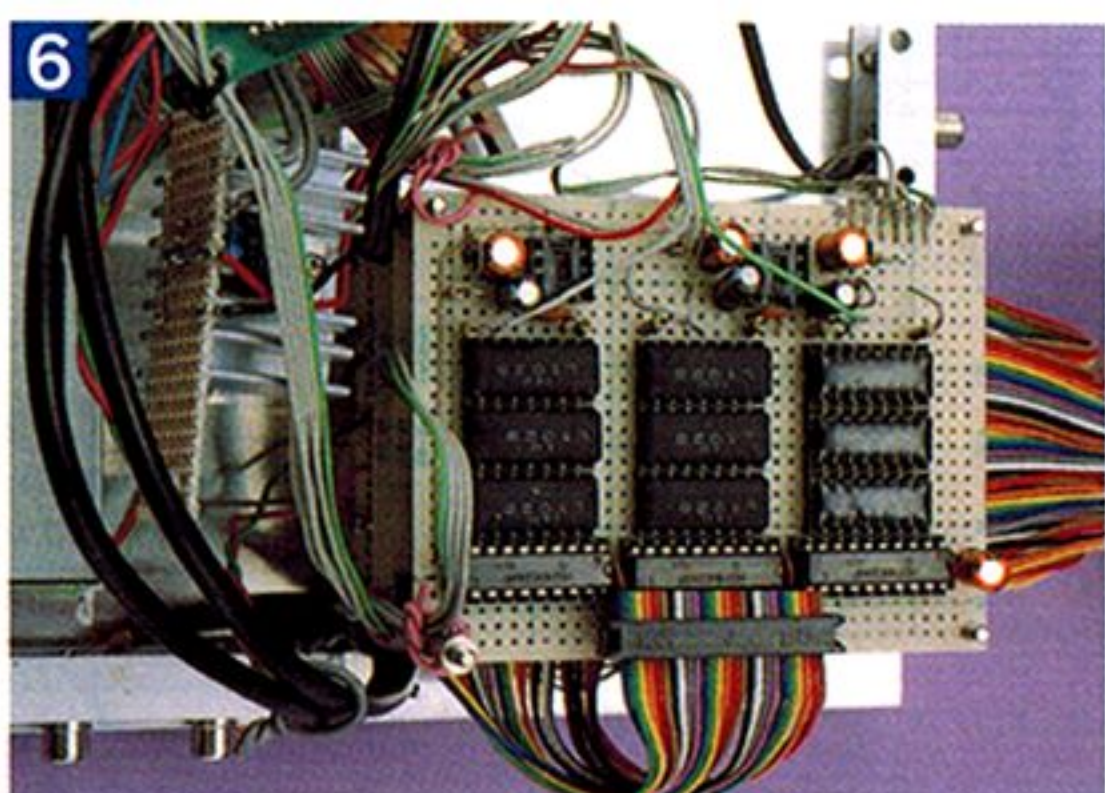
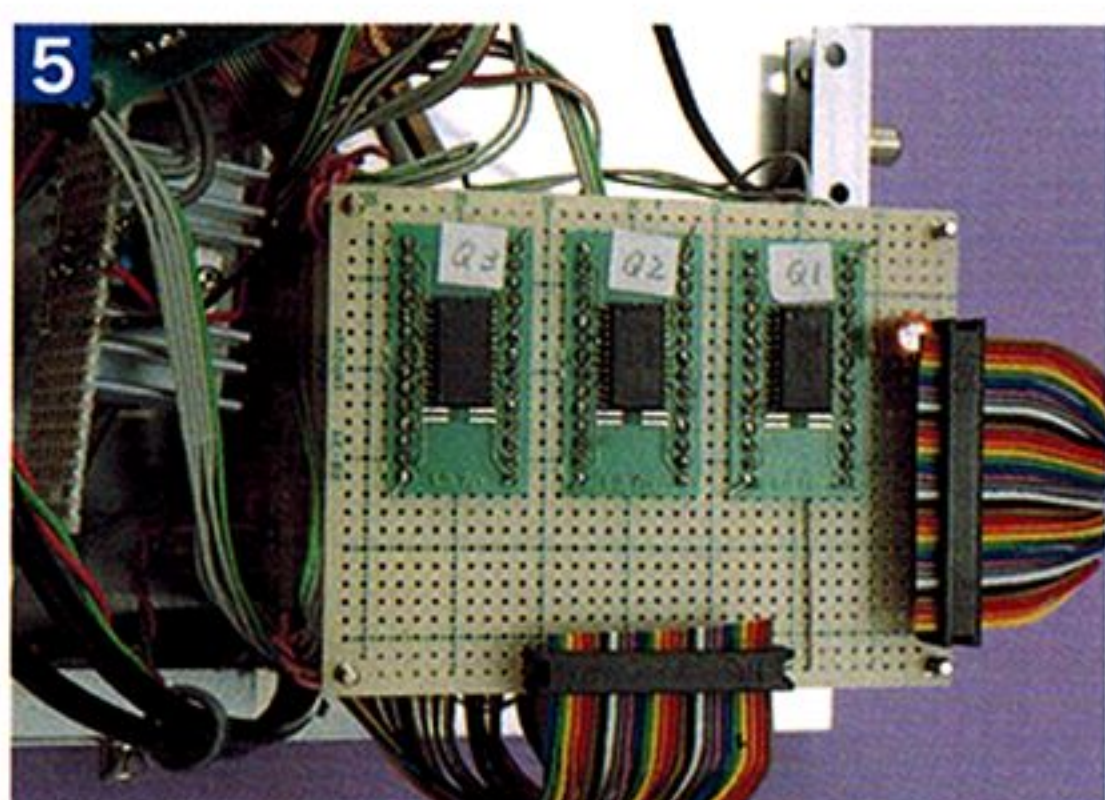
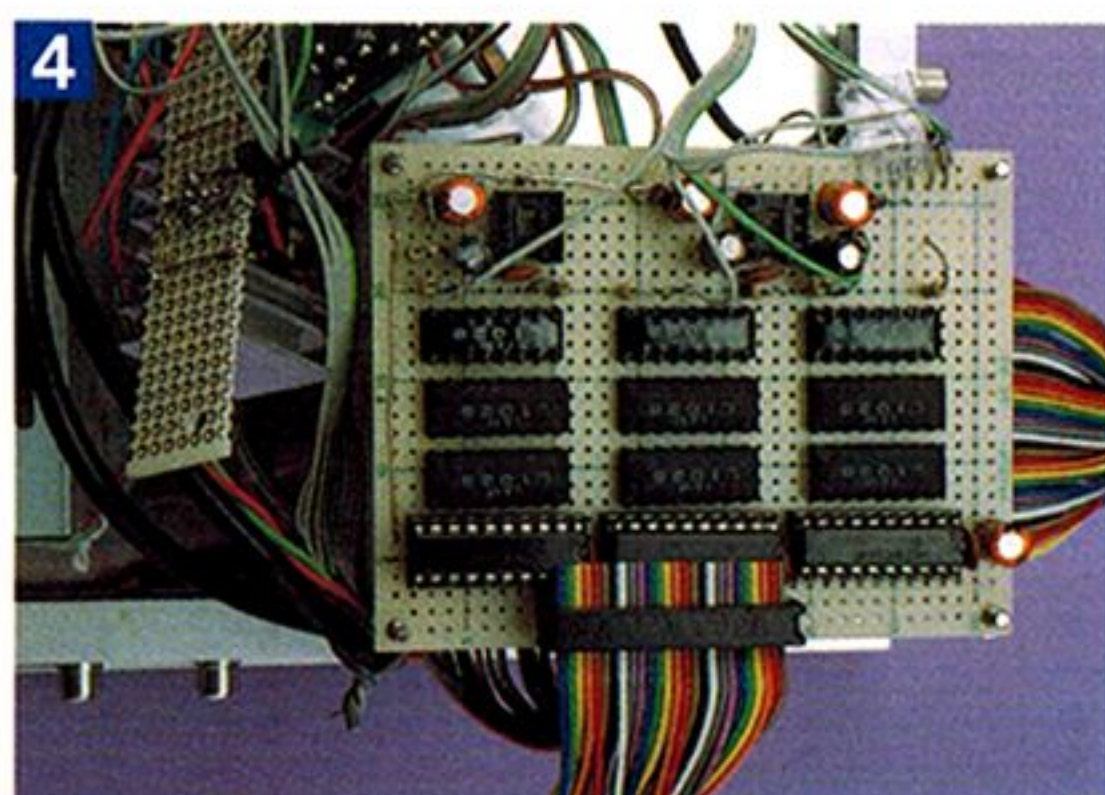
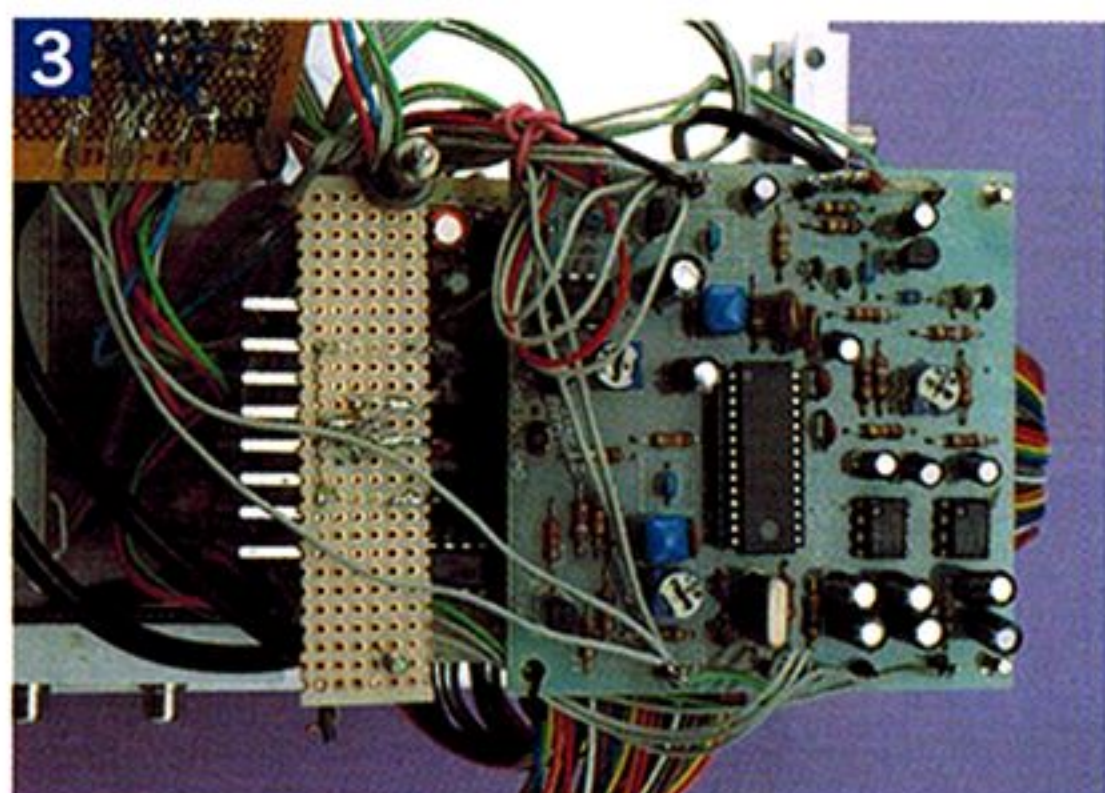
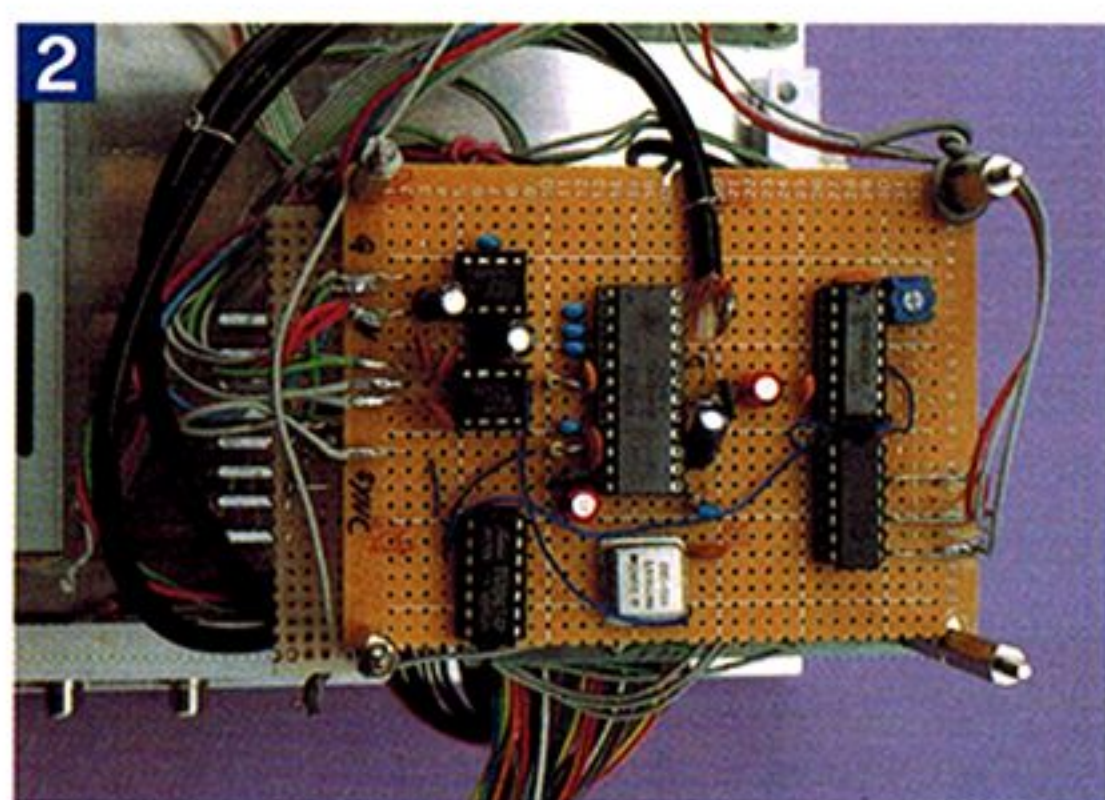
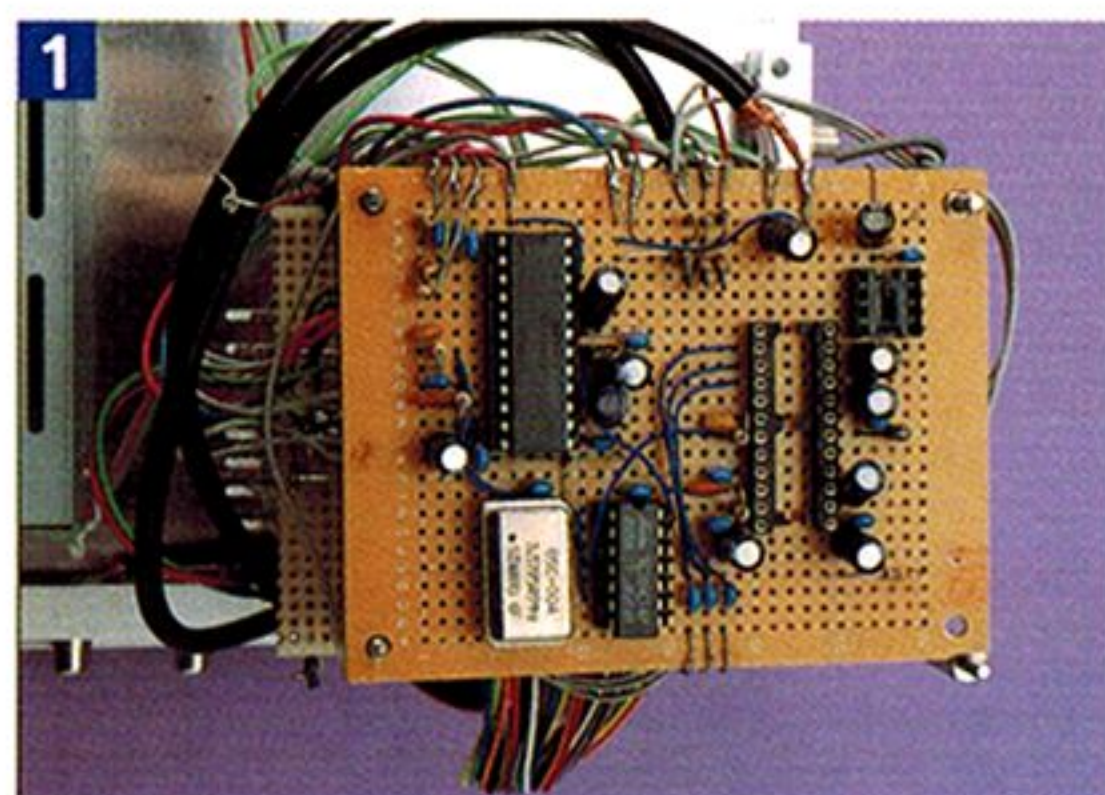
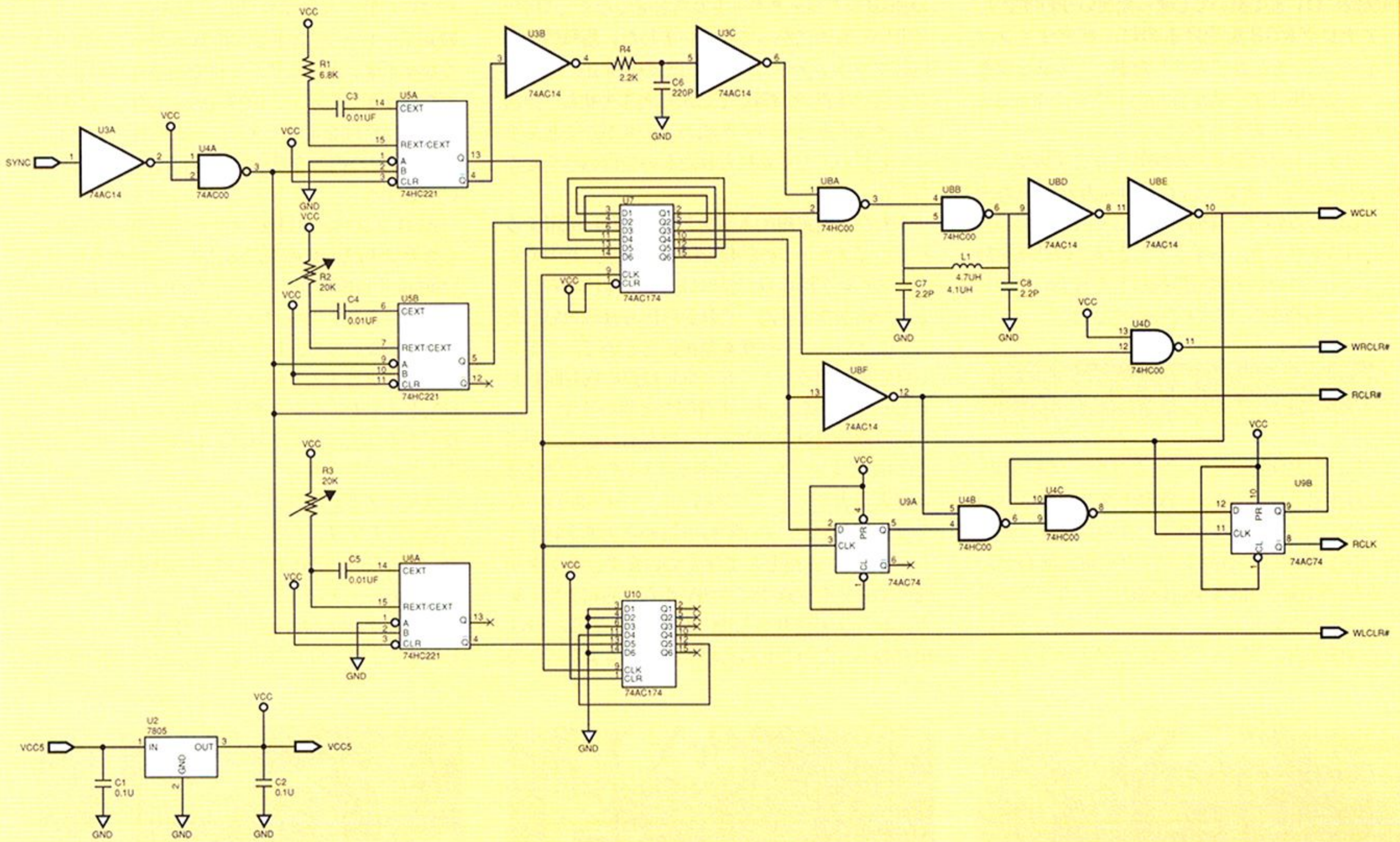
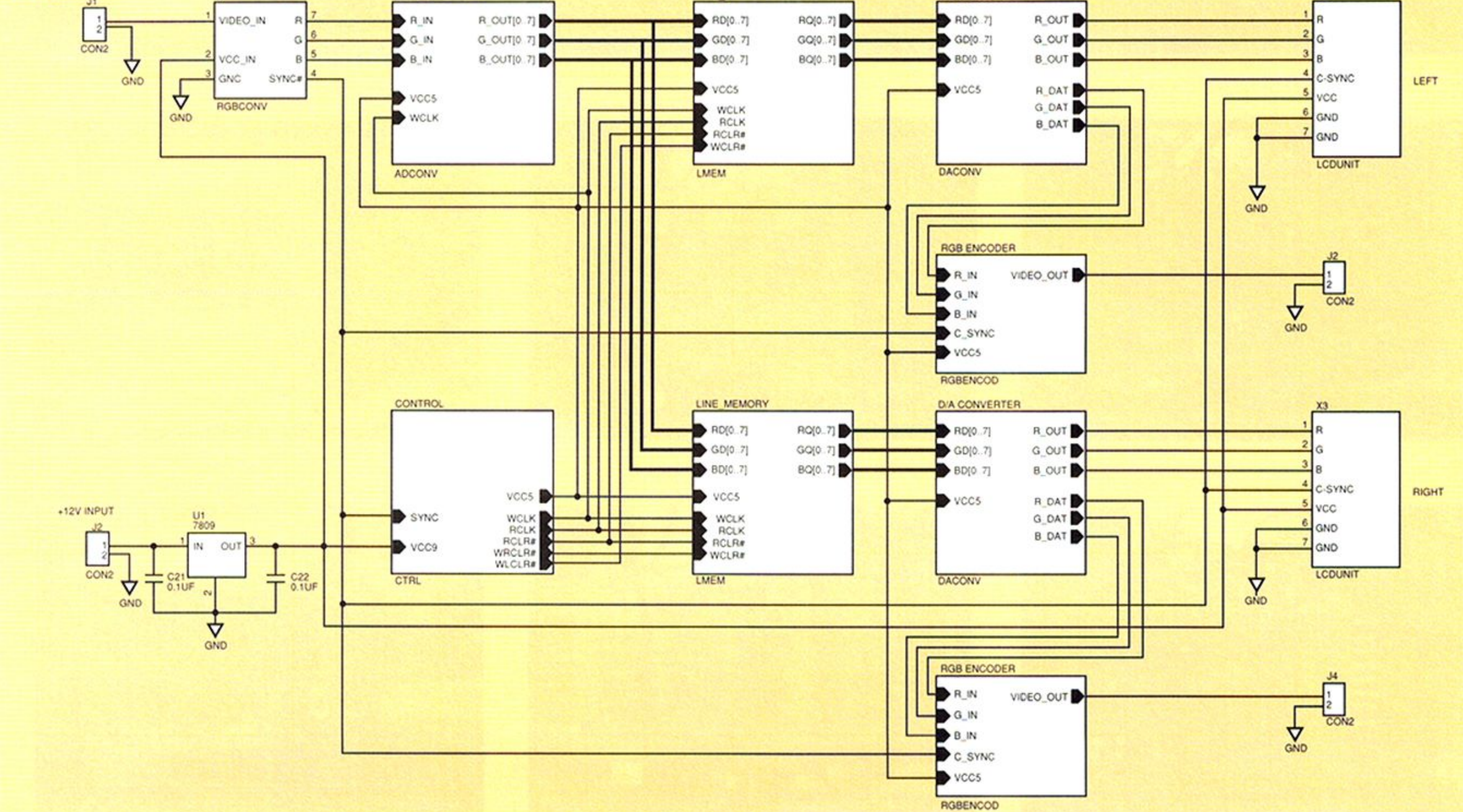



図13

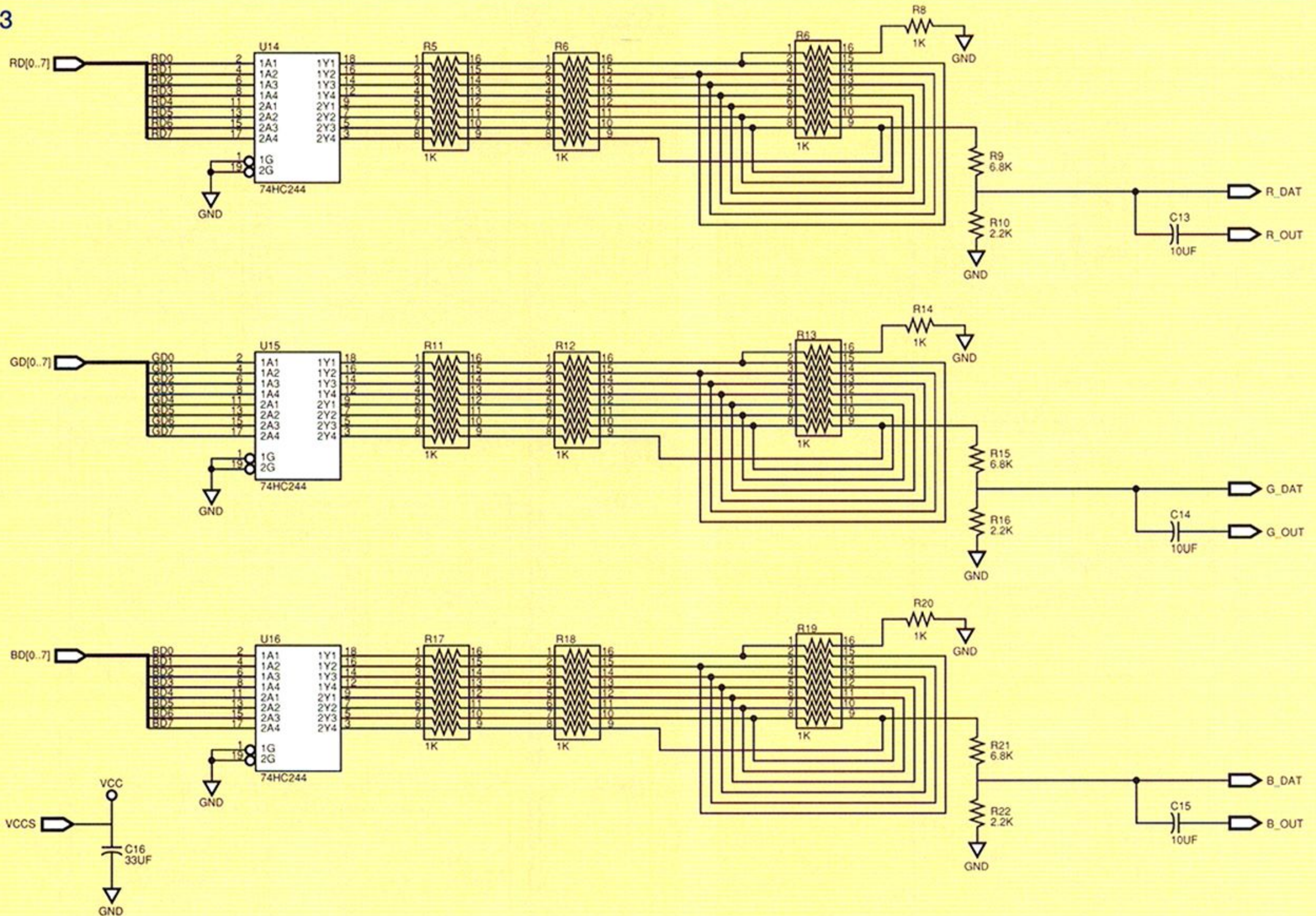



図15

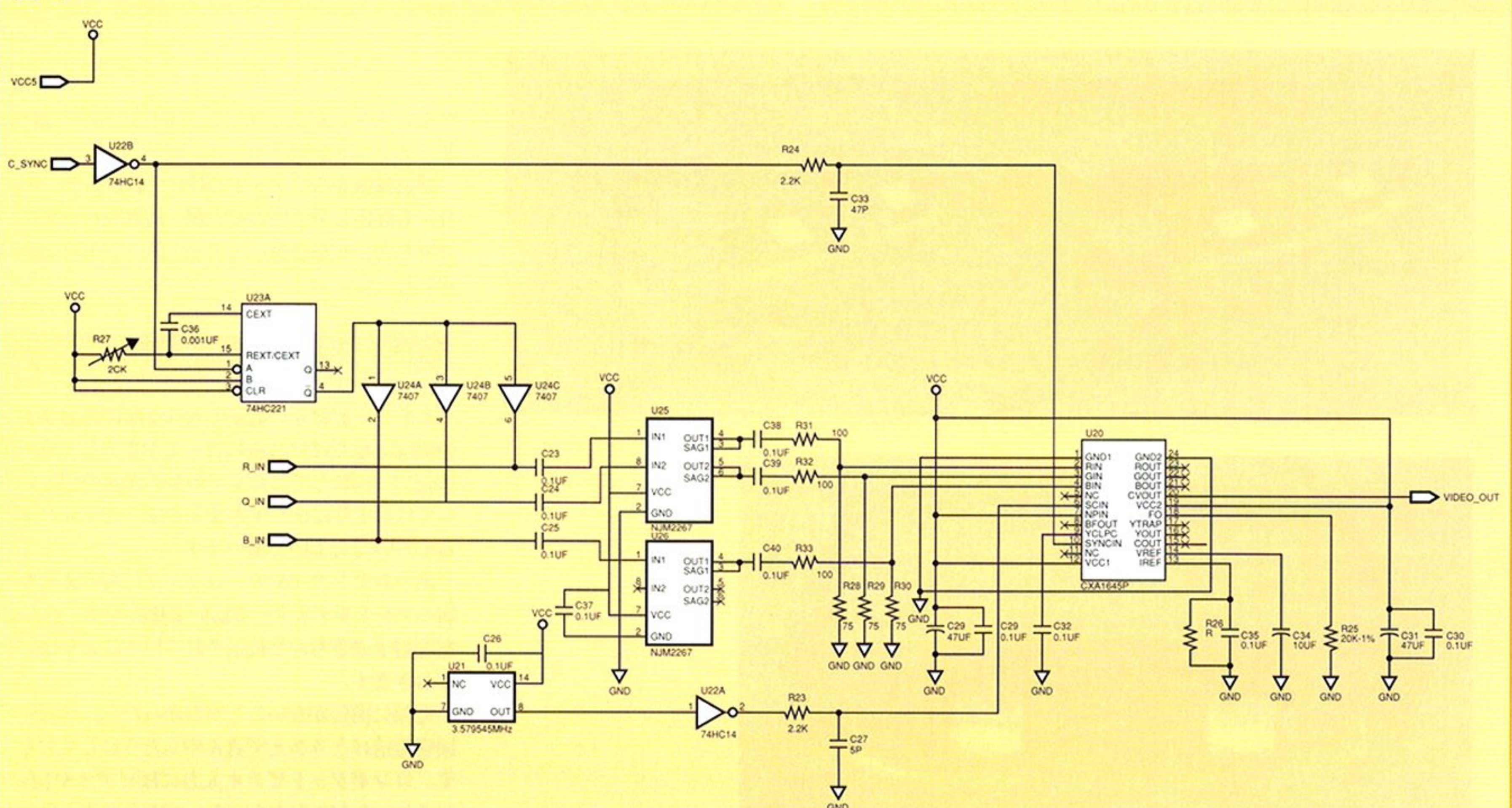



図14

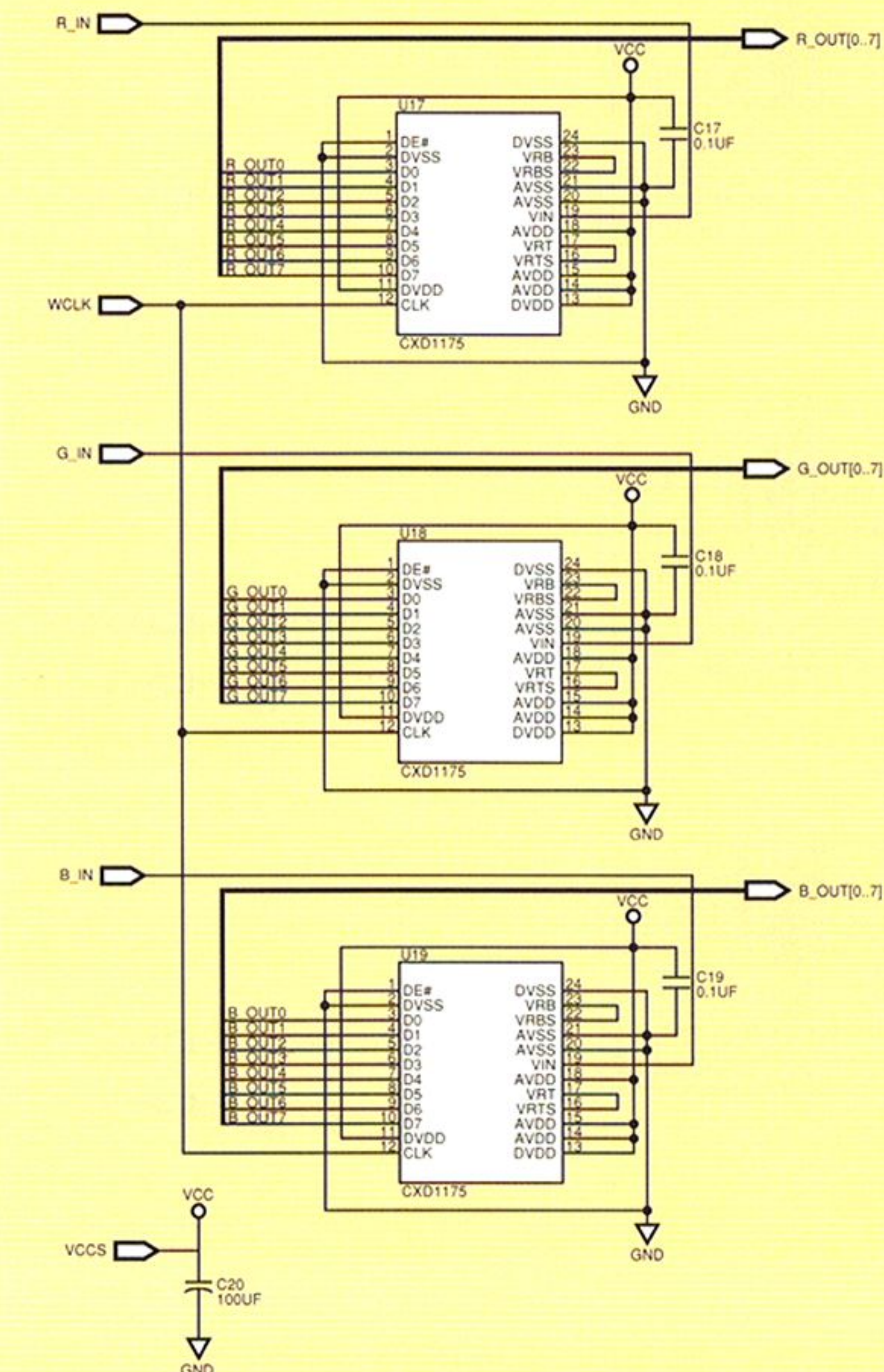
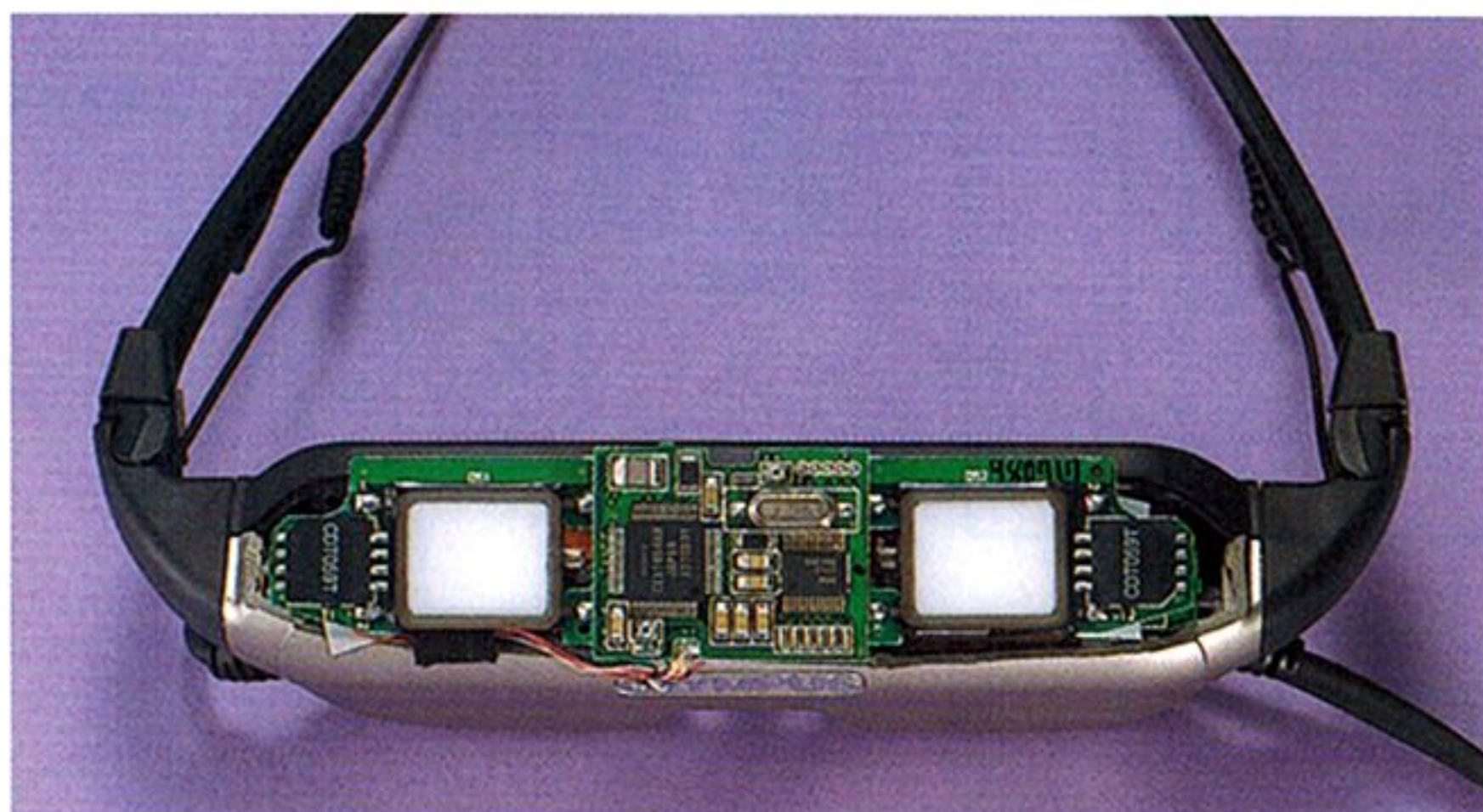
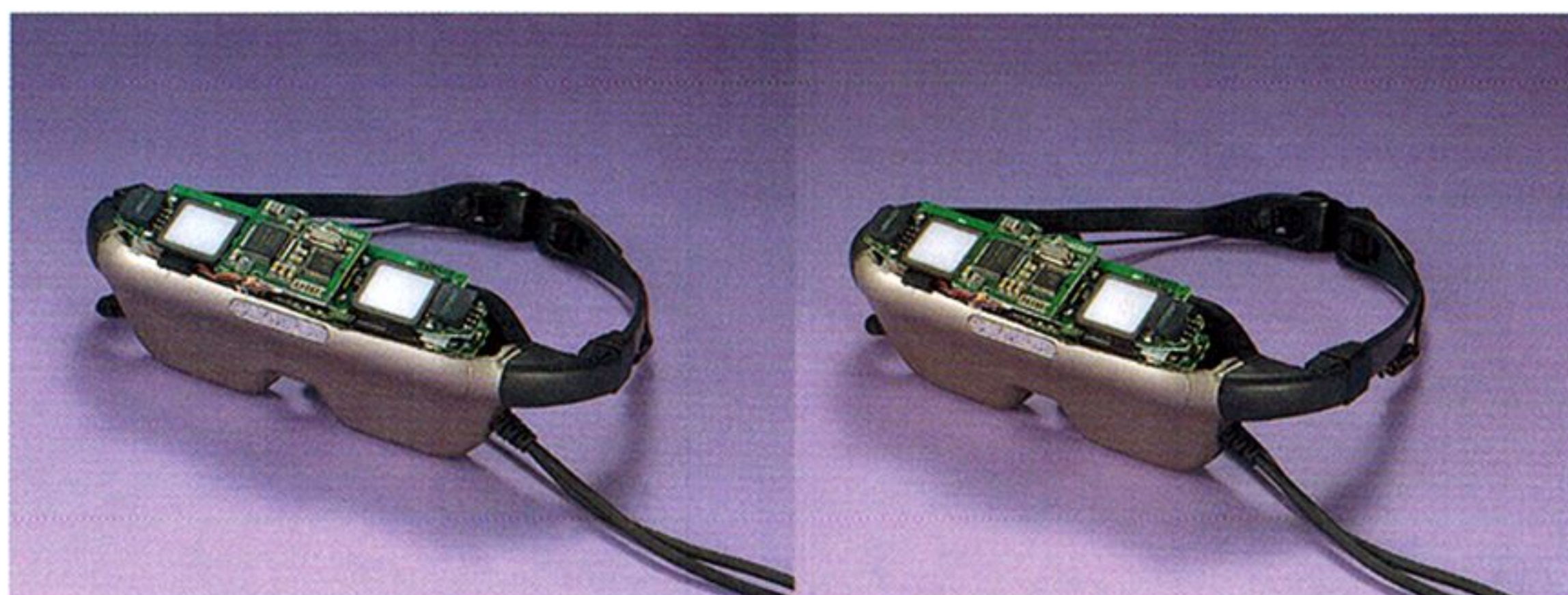
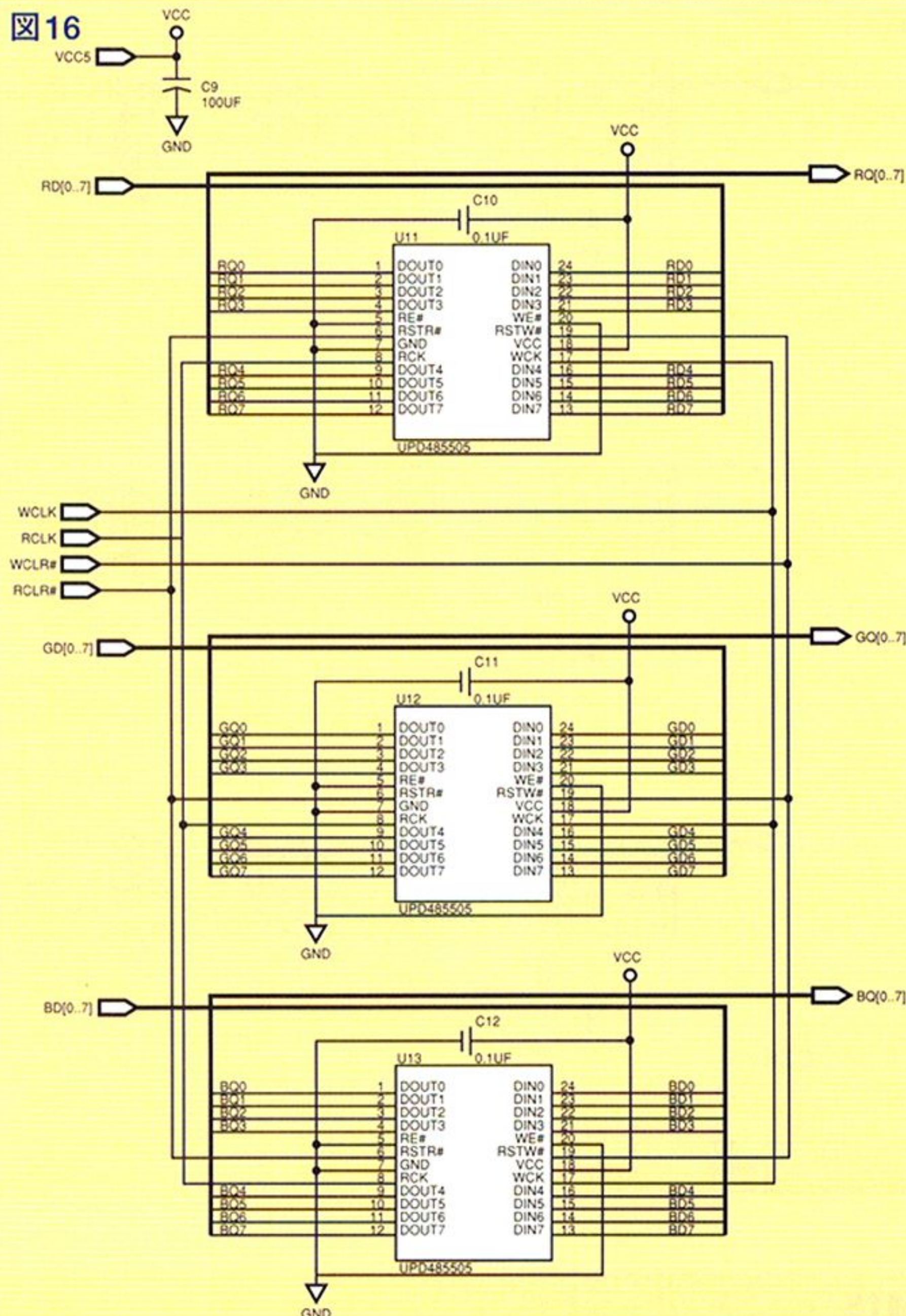



図16

平たいアルミ板などの上で一列をまとめて曲げるとうまくいきます。

変換基板からソケットづけにすれば万が一の場合にも対応しやすいのですが，適当なヘッダピンの持ち合わせがなかったので，錫メッキ線で直付けしました。

ビデオスプリッタの調整

セオリーどおり，電源を入れる前に電源ラインの接続確認だけは厳重に行っておきましょう。逆接続で一瞬にしてこれまでの苦労が水の泡というのではあまりに悲しすぎますので，念を入れすぎるくらいでもよいと思います。

ホームセンターやカーショップにあるような安物のデジタルテスターで構いませんので，各ICの電源端子がきちんと接続されていることを確認しておきます。

電源に問題がないことがわかったら，基板の半固定抵抗はとりあえず真ん中あたりにしておきます。コンポジットビデオ入力にはビデオやLDプレイヤーなどの出力をつないでおきます。私は昔手に入れたテレビの音声多重チューナにビデオ信

1

4

7

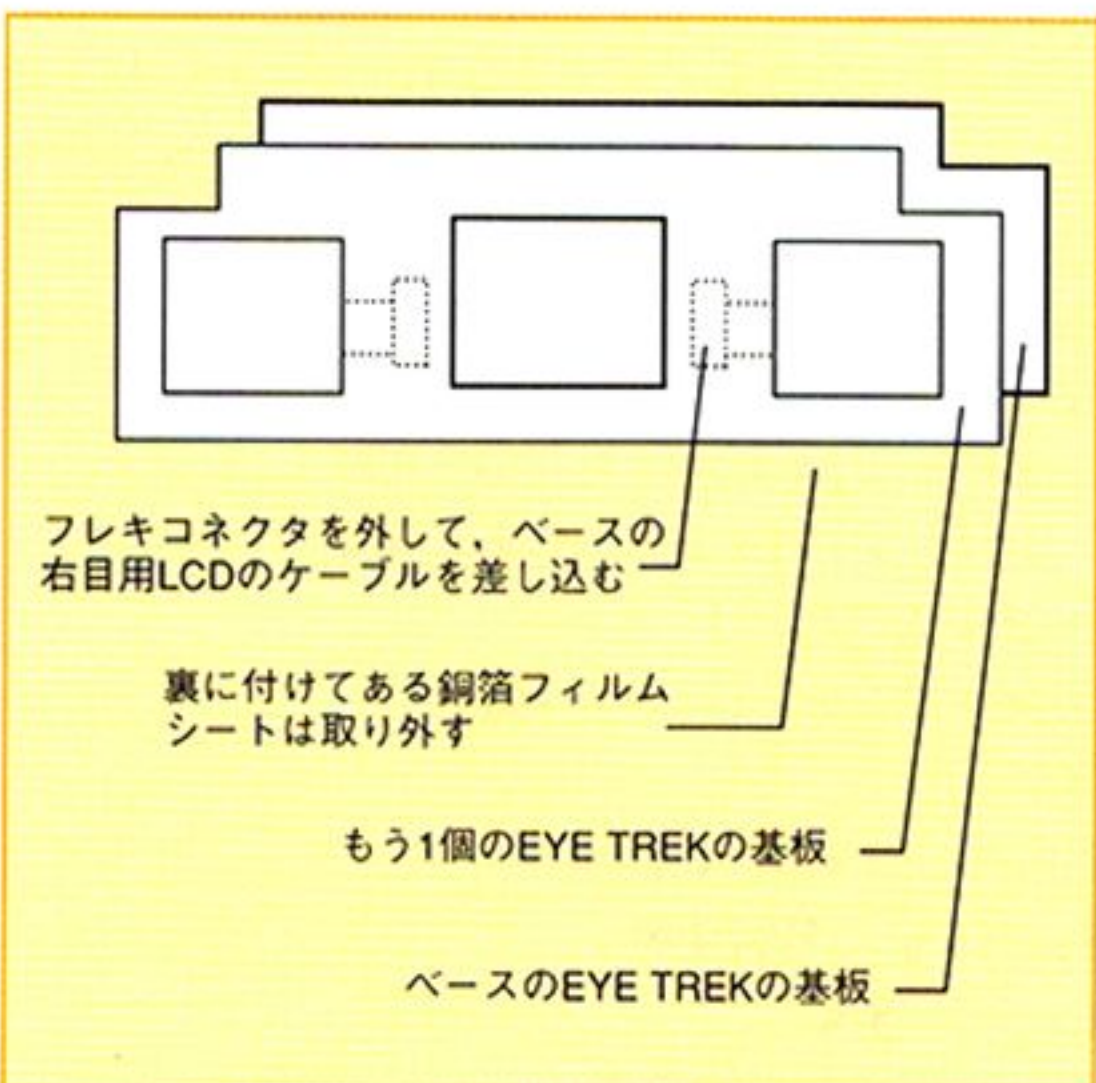
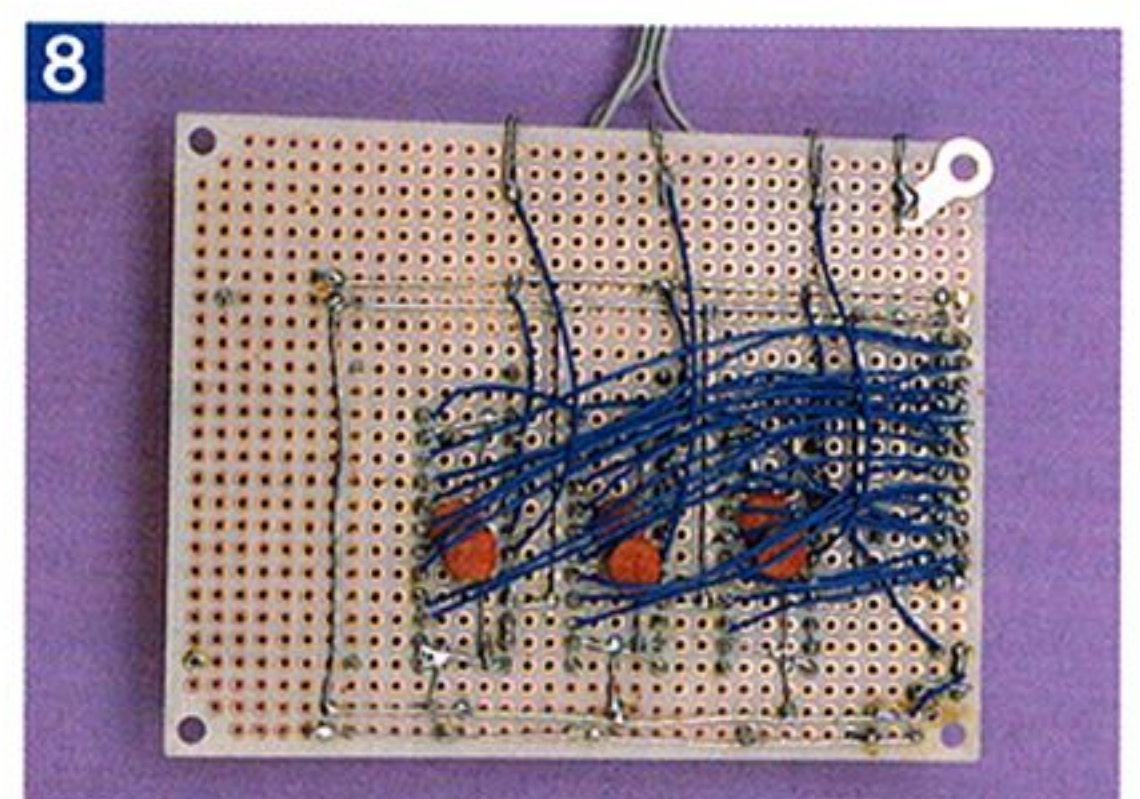
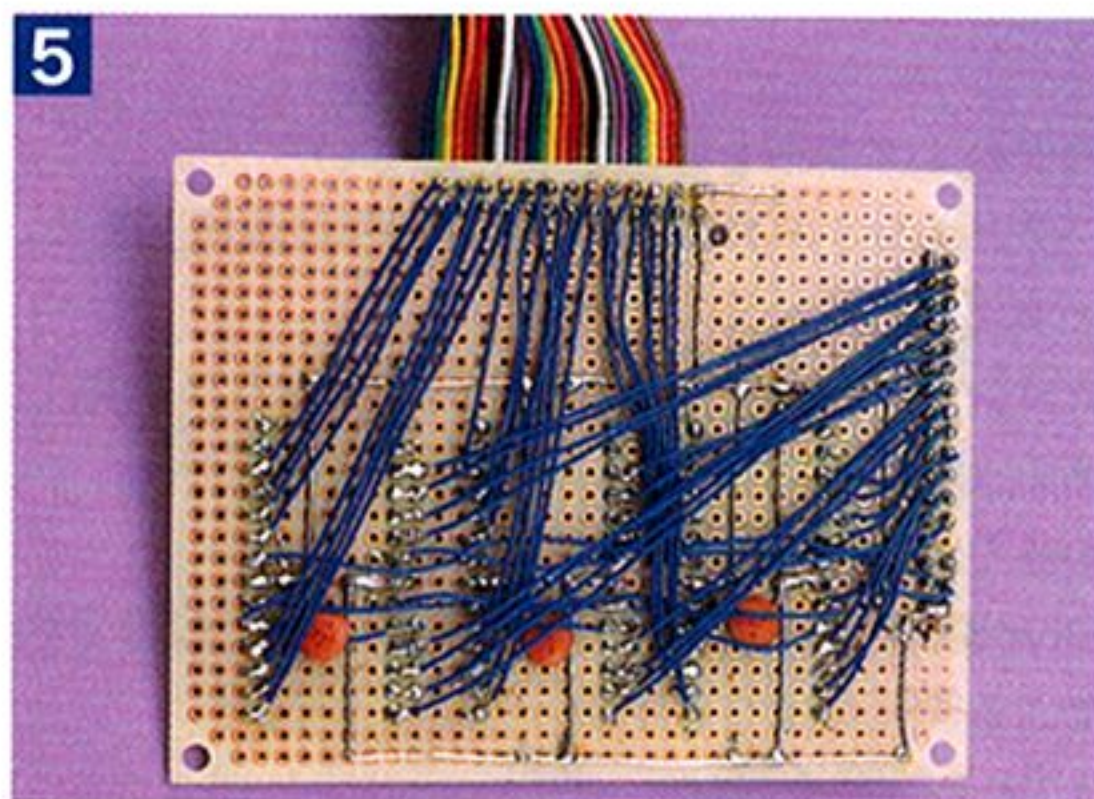
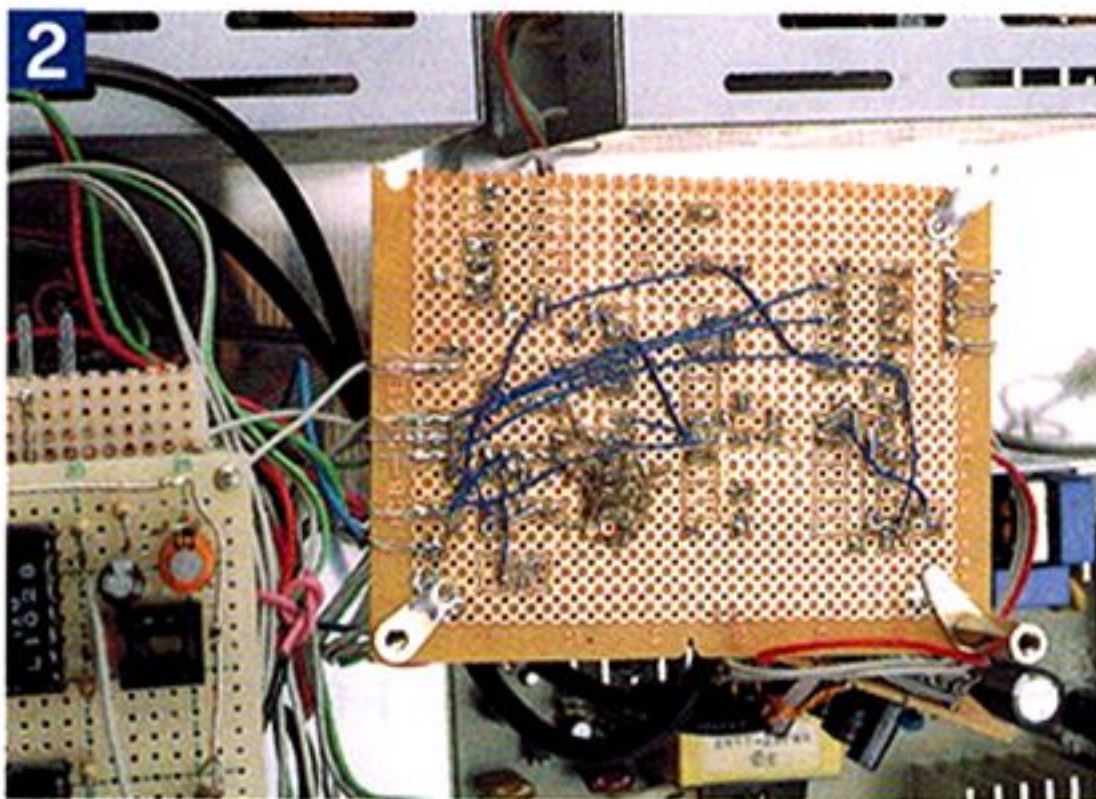
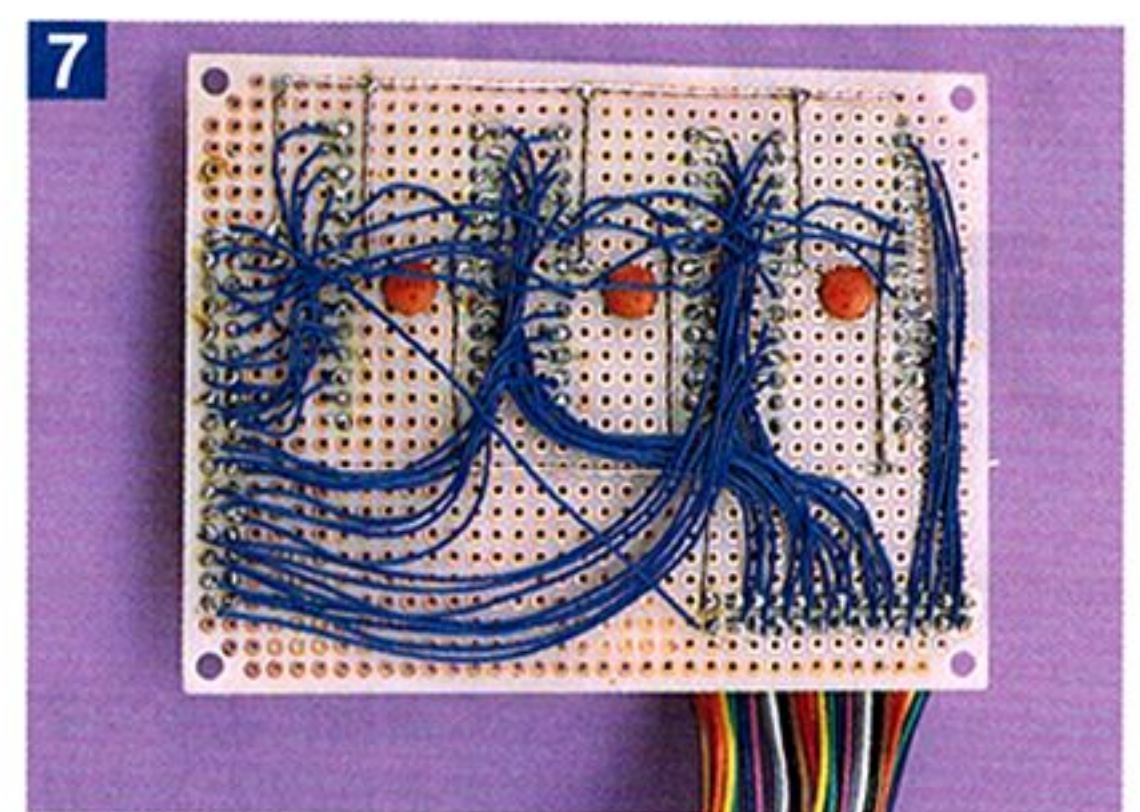
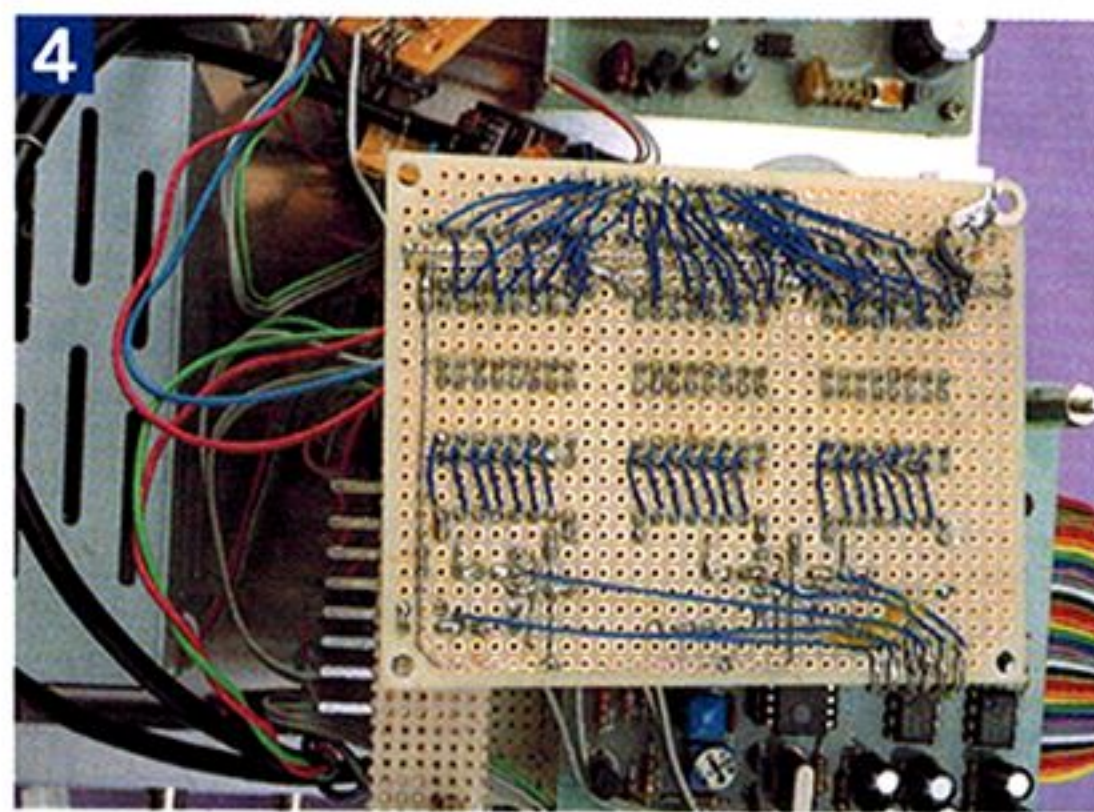
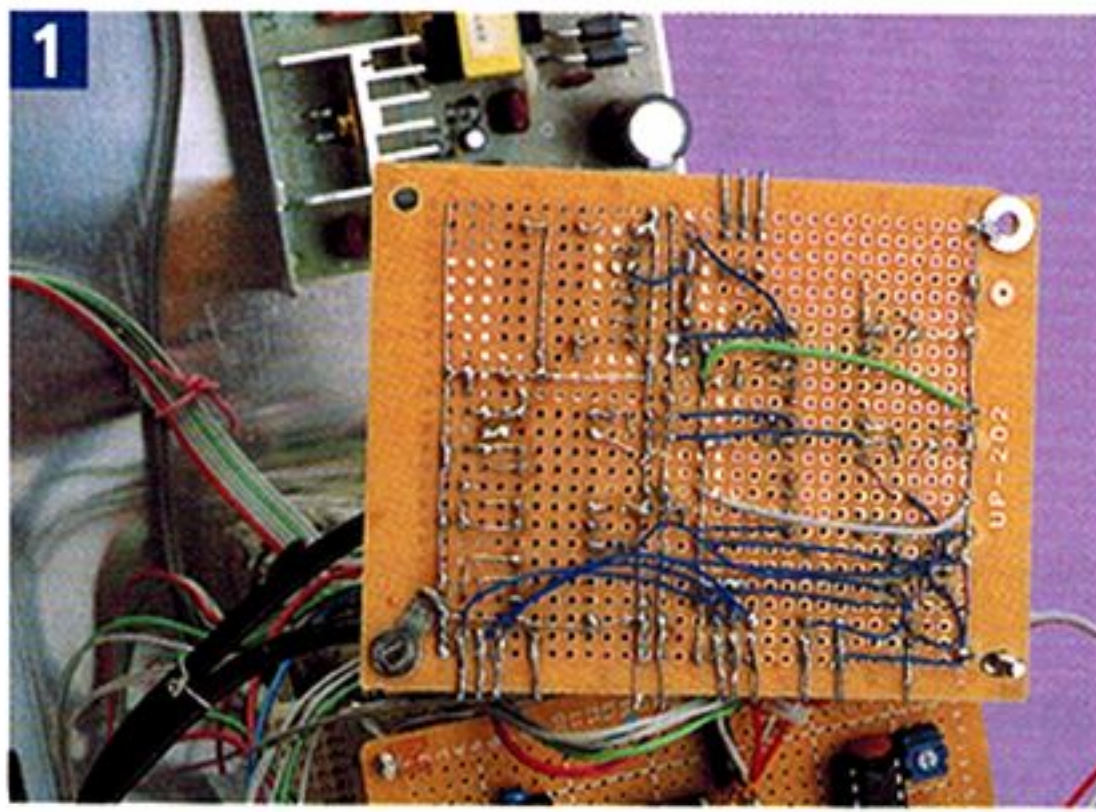
2

5

8

図10　EYETREK基板の増設

6

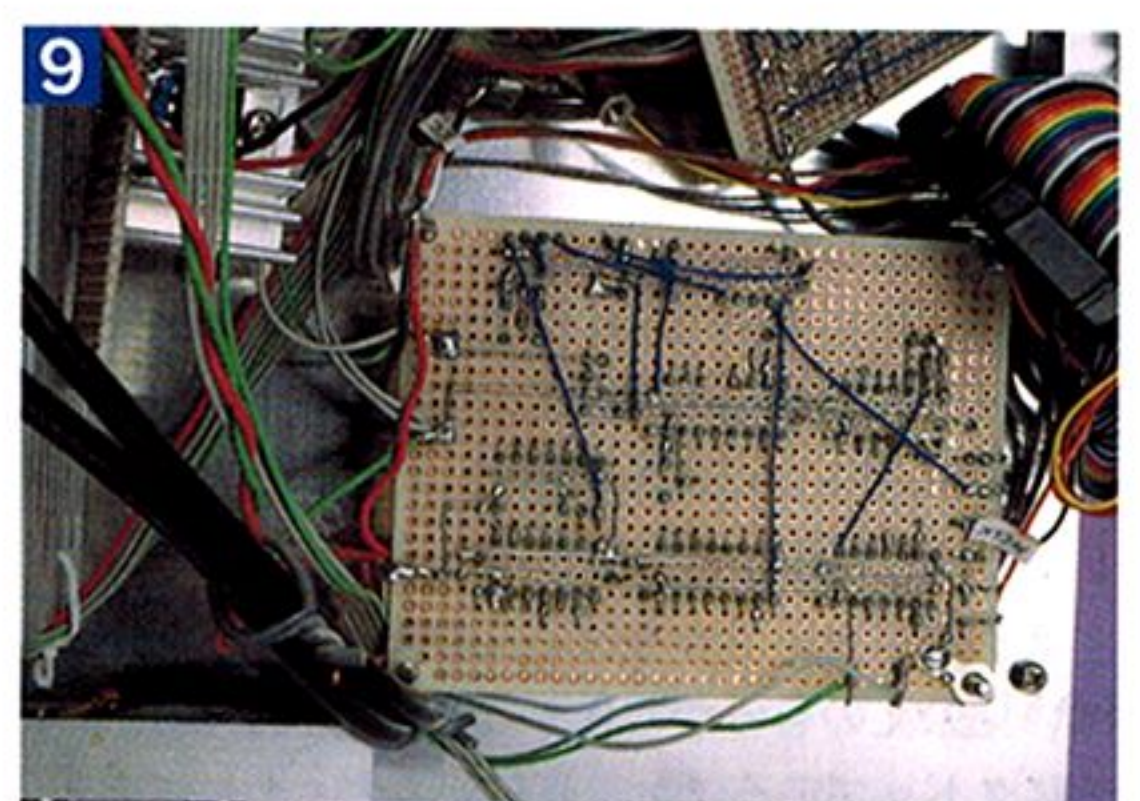
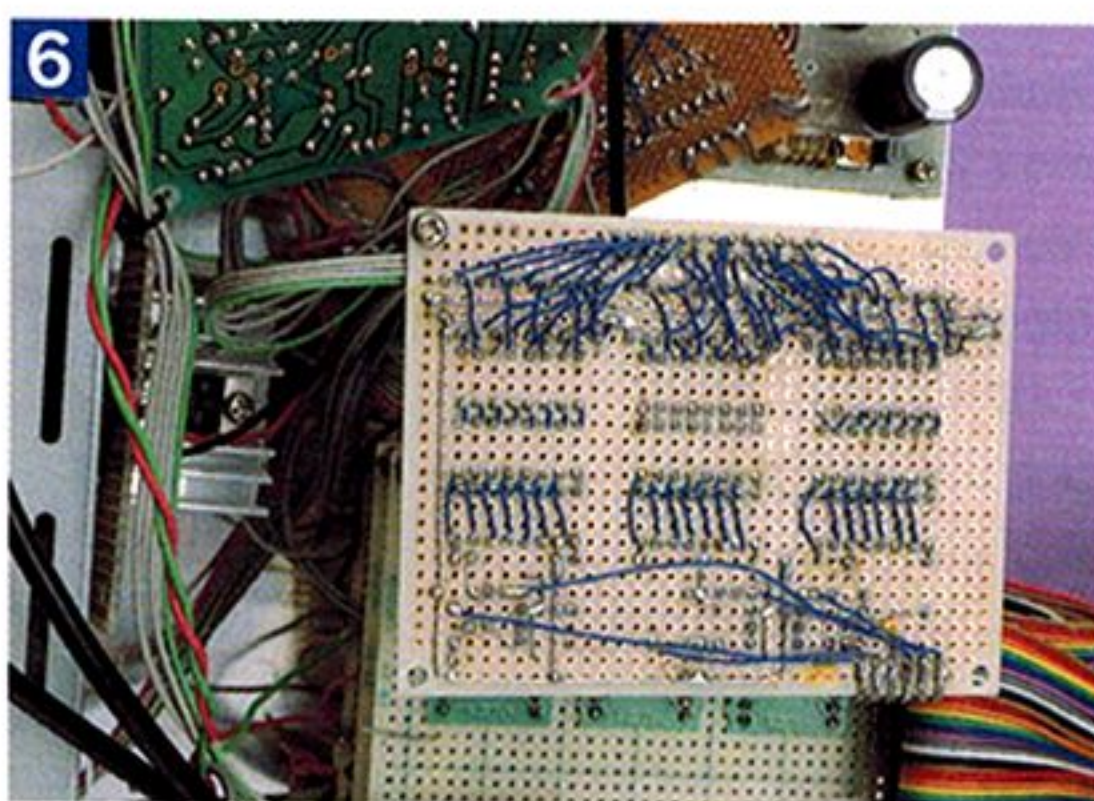
9

号出力があったので，それを使いました。

電源を入れると，液晶ユニットのバックライトが点灯して，なにやら画面が出るはずです。ここで表示位置調整の半固定抵抗を回して右用が画面の右半分，左用が左半分を表示するように設定します。

続いて，RGBコンバータのほうで色合いとコントラストを調整しておきましょう。これでとりあえず完成です。

EYE TREKの改造

さて，当初この製作記事はここで終わりだったのですが，最初にお話ししたとおり，なんとかEYE TREKでやろう！ということで，改造に手を出すことにしました。

まず，EYE TREKを分解してみます。分解する前は，どうせ1系統しかないのだし，液晶パネルはひとつだけで，その画像を光学的に左右に振り分けているのだろうと思っていました。ところが，分解してみると液晶パネルは左右それぞれについています。

さらに幸運なことに，この液晶パネルと基板の接続がコネクタになっています。このコネクタを抜いて，液晶パネルの信号を他から供給するようにすれば左右別々の画像を表示できることになります。

液晶の信号をたどっていくと，ケーブルからもらったビデオ信号を液晶パネル用の信号に変換するIC（SONYのCXA1854）で受けて，このICが2個の液晶パネルを駆動するという形になっています。このIC周辺回路をそのままそっくりいただければよいのですが，映像信号の供給まで考えると自作には少々厳しいものがあります。

そこで，思い切ってEYE TREKをもう1台犠牲にすることにしました。この制御基板をそっくりいただいて，制御基板を2枚持たせてしまおうというわけです。（図10）

フレキを少々強引に曲げれば，なんとかとベースとなるの制御基板の上に載せられるということはわかったので，ケースの上半分を切除して，下のほうにケーブル引き出し口をつけました。これで，ビデオ入力2系統持った立体視対応のヘッドマウントディスプレイが完成です。

試しに通常のビデオ信号を入力して綺麗に映っていることが確認できれば改造は終了です。

おわりに

少々苦労させられましたが，なんとかコンポジットビデオ信号で与えられた画像を左右に分離するビデオスプリッタ，そしてヘッドマウント式の立体視ディスプレイが作成できました。

ただ，試作したものは，やはりアナログ信号を扱うにしてはかなり雑な作り方をしているため，画質もお世辞にもよいとはいえません。できれば専用基板を作るなどして，またアナログとデジタルの電源は綺麗に分離するなどして作成するべきでしょう。

ヘッドマウントディスプレイのほうはEYE-TREKが立体視に対応できるような機構であることがわかったのは大きな収穫でした。ここまでやっているなら，立体視に対応可能な仕掛けを最初から入れてほしかったような気がします。

TkDesktop－Tcl/Tkで作るデスクトップ環境

こて丸　Kotemaru

Tcl/TkやC言語によるコマンドなどを組み合わせて作成されたX Window用の独自デスクトップGUI環境だ。TkによるGUIとスクリプトやコマンドなどの機能追加記述というわかりやすい構成となっており，GUI環境をとことんカスタマイズしたいという向きには参考になるだろう。

開発を始めた動機

いまでこそGNOMEやKDEがもてはやされていますが，私がOh!X休刊に伴いX68000からDOS/V機に乗り換えた頃にはX Window上のGUI環境といえるものは皆無でした。元々UNIXユーザーでもあった私は「kterm4つ開けたらほかに必要ないじゃん」という意見も理解できたのですが，すでにSX-Windowにどっぷりと浸かっていたのでどーしてもGUI環境がほしくなっていました。当初，アテナウィジェットをポチポチといじっていたのですがGUI環境を構築するには膨大なコーディングが必要そうで途方にくれていました。

そんなとき「わずか10行でらくらくGUIプログラミング」などという満開製作所の広告のようなコピーのついた書籍を見つけました。これがTcl/Tkとの出合いです。

最初にファイラーを作ってみたところ，それっぽい見栄えのものができてしまい，調子にのってデスクトップアイコン，そしてエディタと作成し，そのあと自分で使いながらこまごまと拡張を続けてきました。そうしてできたのがTkDesktop環境です。

概要

TkDesktopはX Window上にTcl/Tkで構築されたデスクトップ環境です。基本的には開発支援環境として構築しました。

ソースファイルのアイコンをクリックするとエディタが起動し編集後，Makefileをクリックするとコンパイルが行われ，そのログからタグジャンプで再びエディタへ……というような流れを実現するものです。

構成と簡単な使い方

TkDesktopは大きく分けて3つのプログラムがあります。

xicons：デスクトップアイコン

デスクトップ上にアイコンを置くプログラムでいちばん最初に起動します。アイコンは定義ファイルによってイメージ，コマンド，メニューなどが定義されます。定義ファイルの詳細はxiconsのドキュメントを参照してください

デフォルトのアイコン定義を簡単に説明します。

・Xロゴアイコン

コマンド：~/.TkDesktopのディレクトリを開く
メニュー：
Exit：xiconsの終了
Save&Exit：環境を保存して終了。次に起動されたとき復元しようとする

・ルートアイコン

コマンド：/のディレクトリを開く
メニュー：
Open：コマンドに同じ
Info /：/デバイスのディスク情報
Info /usr：/usrデバイスのディスク情報
Info /var：/varデバイスのディスク情報

・ホームアイコン

コマンド: $HOMのディレクトリを開く
メニュー：
Open：コマンドに同じ
Info：/homeデバイスのディスク情報

・CD-ROMアイコン

コマンド：/cdromのディレクトリを開く
メニュー：
Open：コマンドに同じ
Eject：umount /cdrom
Info：/cdromデバイスのディスク情報

・FDアイコン

コマンド：/FDのディレクトリを開く
メニュー：
Open：コマンドに同じ
Eject：umount /FD
Info：/FDデバイスのディスク情報

・時計アイコン

コマンド：カレンダーウィンドウを開く
メニュー：なし

・メールアイコン

コマンド：なし
メニュー：なし

・プリンタアイコン

コマンド：プリントダイアログを開く
メニュー：なし
ドロップ：プリントダイアログを開く

・端末アイコン

コマンド：ktermを起動する
メニュー：なし

xdir：ファイラー

TkDesktopの中心となるプログラムです。特定のディレクトリ配下に存在するファイルをアイコン化します。通常はデスクトップアイコンから起動されます。

アイコンは定義ファイルによってイメージ，コマンド，メニューなどが定義されます。定義ファイルの詳細はxdirのドキュメントを参照してください。

xdirからxdirにアイコンをドラッグ＆ドロップすることでファイルの複製，移動，シンボリックリンクを行うことができます。ファイル入力項目にアイコンをドロップすればファイル名がフルパスで入力されます。選択したアイコンのフルパスはセレクションに設定されるのでktermなどでは中ボタンを使ってください。

※制限事項：空白や$を含むファイル名は誤動作します。

デフォルトのアイコン定義を簡単に説明します。

・ディレクトリ

コマンド：ディレクトリ移動
メニュー：
Open：コマンドに同じ
Rename：名前変更
Copy：複製
Attribute：属性の表示

・実行ファイル

コマンド：自分自身を実行
メニュー：
Edit：xedで開く
Vi：viで開く
Rename：名前変更
Copy：複製
Attribute：属性の表示

・テキストファイル

コマンド：xedで開く
メニュー：
Edit：xedで開く
Vi：viで開く
Rename：名前変更
Copy：複製
Attribute：属性の表示

・その他

その他，適当に定義されています。定義ファイルを参照してください。

※CTRL+右ボタンでEditIconというメニューが出て，GUIからアイコンを編集できます。ただし仮実装のものです。

xedit：エディタ

シンプルなテキストエディタです。Tcl/Tkのtext機能を使用しているので基本的にそれ以上の機能はありません。

特殊な機能として標準入力をエディタへの入力とすることができます。これはCUIのプログラムの出力をログとして得るときに使います。

プログラムについて

TkDesktop はたくさんの独立したモジュールから構成されています。

それぞれのモジュールはwish8.0jpのスクリプトとして記述されています。Tcl/Tkで記述しきれない部分はC言語で記述されており，それらは独立したコマンドとなります。

このモジュール群は基本的に1モジュール1プロセスとなります。メモリ効率は悪いですがプログラムがコケたときに全体をまきぞえにしないなど，よい点もあります。また，Tcl/Tkはグローバル変数を使わないとなにもできないためプログラミングの負荷を増やさないという意味もあります。

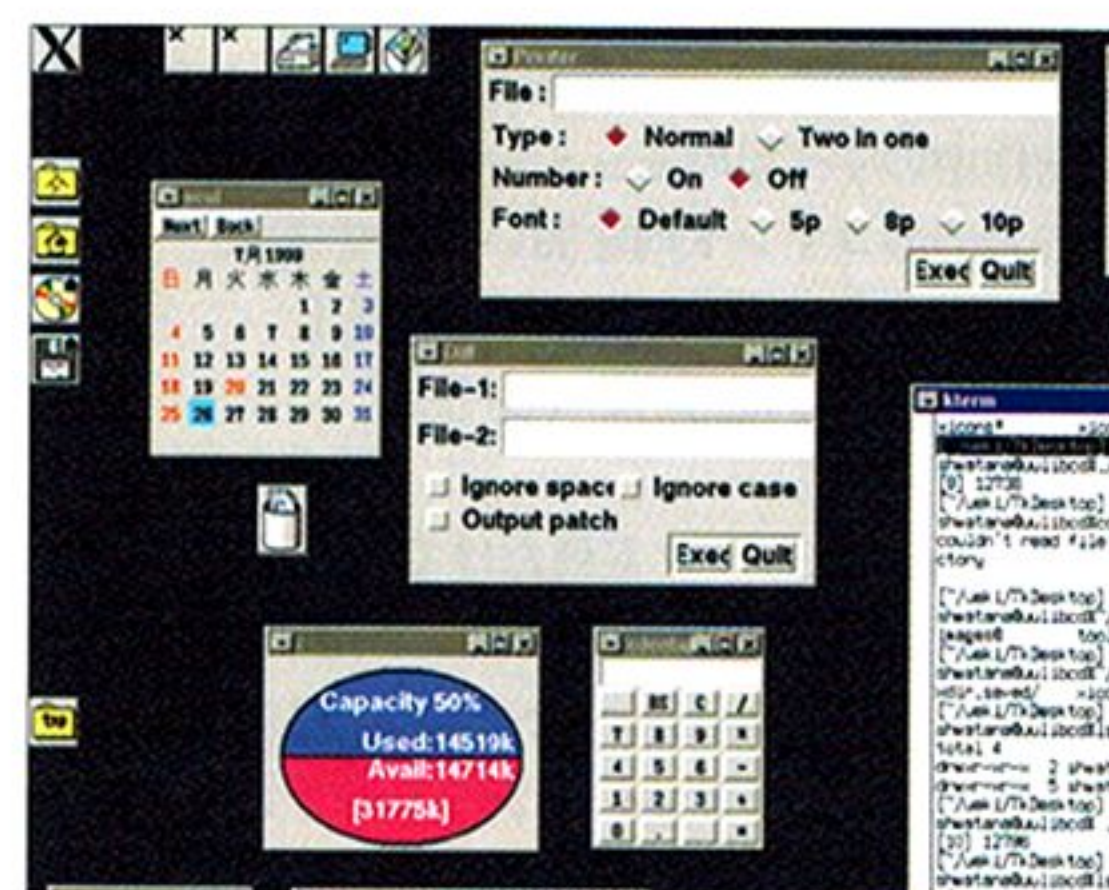
各モジュールの簡単な説明をします。

・chai
画面中央にメッセージを表示する。通常時計アイコンのスケジュール機能から呼び出される。
・clip_board
カット＆ペーストのバッファ。常駐する。
・printer
プリントダイアログ。a2ps を使用。
・xattr
ファイル情報の表示。パーミッションの変更機能も持つ。
・xcal
カレンダー。
・xdentaku
電卓。
・xdiff
diff コマンドのGUI ラッパー。結果はxed で表示される。
・xdir
ファイラー。
・xdiskinfo
ディスク情報の表示。df コマンドのGUI ラッパー。
・xed
テキストエディタ。
・xgrep
grep コマンドのGUI ラッパー。結果はxed で表示される。
・xicons
デスクトップアイコン。
・xtrash
ゴミ箱。ファイルをドロップされると起動時のカレントディレクトリへファイルを移動する。
・src/copydu.c
コピーするファイルのサイズを事前に調べる。
・src/hash.c
引数で指定された文字列から10桁のハッシュ値を返す。
・src/list.c
専用 ls コマンド。xdir が要求する特殊なソートを行う。
・src/symln.c
専用 ln コマンド。絶対パスで与えられたリンク先を相対パスに変換してシンボリックリンクする。
・src/copy.c
専用 cp,mv コマンド。転送先に存在するファイルのみや更新されているファイルのみを対象とするオプションを持っている。
・src/base64.c
base64 エンコーダ。Tcl/Tkのimage コマンドの -data オプションがbase64化データを要求するためのフィルタ。
・parts/MENU.tcl
メニュー定義コマンド。定義ファイルで使用。
・parts/Buttons.tcl
複数のボタンを簡単に記述するためのパーツ。
・parts/TEntry.tcl
タイトル付きEntry パーツ。アイコンのドロップとカット＆ペーストに対応している。
・parts/Radios.tcl
複数のラジオボタンを簡単に記述するためのパーツ。
・parts/One_Name.tcl
ひとつのTEntry とExec，Quitの2つのボタンからなるダイアログ。ファイル名の指定などに使う。
・parts/Image.tcl
イメージをロードするプロシージャ。
・parts/Copying.tcl
ファイルの複写中用のダイアログ。
・parts/Dialog.tcl
汎用のダイアログ。
・parts/ALL.tcl
すべてのパーツを読み込む。
・xicons.parts/clock.tcl
xicons 用の時計アイコン。
・xicons.parts/mailbox.tcl
xicons 用のメールアイコン。

いくつものモジュールが別プロセスとなるのでプロセス間通信を行っています。プロセス間通信にはTcl/Tkのsend コマンドとセレクションを使用します。

モジュールを追加する場合のインタフェイスを説明します。

アイコンのドロップ

アイコンをドラッグ＆ドロップした場合，マウスのボタンを離したときのイベントはボタンを押し始めたウィンドウへ上がってしまいます。そのためドロップを起こしたモジュールはほかのモジュールへ向かってブロードキャストします。

ブロードキャストで問い合わせを受けたモジュールはマウスカーソルが自分のウィンドウ上にあればドロップを受け付けます。このとき，ドロップされたファイル名はセレクションに入っています。複数のアイコンがドロップされた場合，空白で区切られます。

問い合わせを受け付けるプロシージャは以下の形式を持ちます。

```
proc Drop_Event { mode RX RY }
```

mode は"copy"，"move"，"link"のいずれかです。RX，RY はマウスカーソルのルートウインドウ上での座標です。

xdir が問い合わせを行うのはグローバル変数SEND_DROP に名前の登録されているモジュールだけです。SEND_DROP は定義ファイルで設定できます。

カット＆ペースト

カット＆ペーストのバッファを利用したい場合，常駐している clip_board のプロシージャを呼び出します。

```
proc Set_CLIP { win }
proc Cut_CLIP { win text }
proc Get_CLIP { }
```

Set_CLIP はバッファへセレクションのデータを設定します。Cut_CLIP はデータを設定後，呼び出し元を呼び出し選択されている領域を削除させます。Get_CLIP は設定されているデータをセレクションに複写します。win は呼び出し元のウィンドウ名，text はテキストウィジェットです。

保存と終了

xicons は起動時に前回の終了時の状態を復元

図1　セットアップ画面

図2　さまざまなツールを呼び出せる

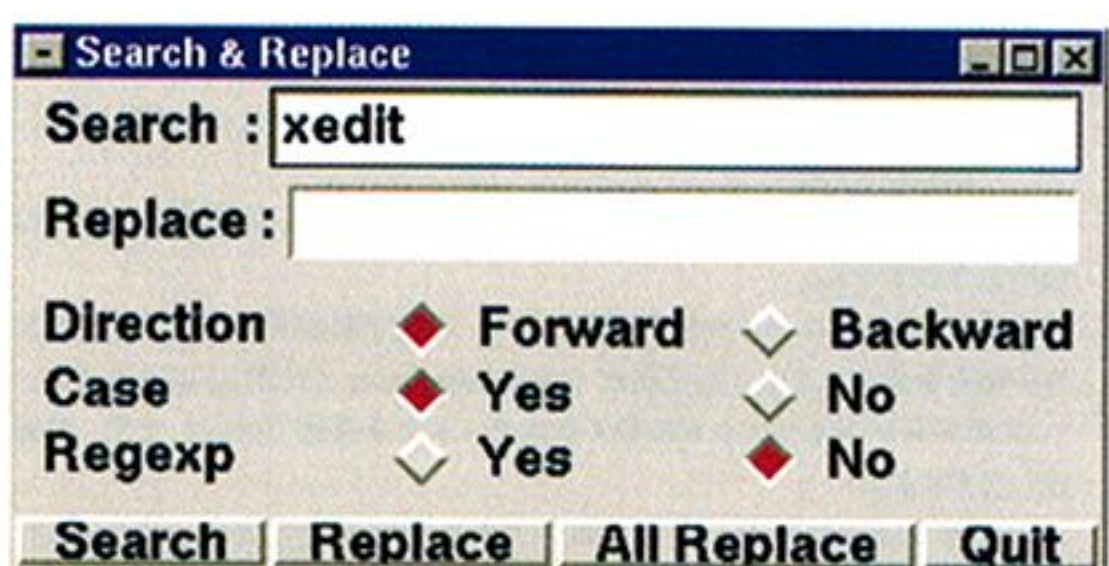



図3

する機能を持ちます。そのためxiconsは終了時に動作中のすべてのモジュールに対して次回再起動のためのコマンド列を問い合わせます。

```
proc Save_This { save_dir } {
return "再起動用コマンド列"
}
```

save_dirはコマンド列だけでは復元しきれない情報を保存するディレクトリが指示されます。通常は ~/.TkDesktop/xicons.savedです。

状態を保存後 xicons はすべてのモジュールに終了通知を出します。

```
proc Exit_Event {}
```

通常はウィンドウマネージャからの終了通知と同じ処理になるので以下のようになっているはずです。

```
wm protocol . WM_DELETE_WINDOW {
Exit_Event }
```

ウィンドウマネージャ

ウィンドウマネージャは基本的にfvwm95を使用しますがほかのウィンドウマネージャでもデスクトップアイコン用にタイトルバーと外枠をなくせるものならなんでも構いません。

リスト1に fvwm95 初期化ファイルのサンプルを示します。

デバイスのマウント

UNIXマシンをクライアントとして使用する場合，スーパーユーザーにならないとデバイスをマウントできないという問題があります。この問題を解決する方法としてユーザーからのアクセス要求に呼応してデバイスをマウントするデーモンを立ち上げておきます。

このようなデーモンにはパッケージとしてすでにamdが存在します。TkDesktopでは専用のデーモンも用意しました。amdを使用することも可能ですがデフォルトの定義ファイルでは専用のものを使うようになっています。

なぜ専用のデーモンを作ったかというと，オートイジェクトを行いたかったためです。しかし残念ながらこの機能は実装されていません。デバイスごとに動作がまちまちで実用に耐えないためです。

専用のデーモンはmountsvrといいます。mountsvrは独立したツールとして提供しています。詳細についてはそちらのドキュメントを参照してください。

インストール

このプログラムはFreeBSD-2.2.7でのみ動作確認してあります。X Window上であればほかのOSでも多分動くと思います。

インストールの説明もFreeBSD-2.2.7を前提とします。

必要なツール類のインストール

FreeBSDのパッケージの入ったCD-ROMをマウントして必要なツール類をインストールします。

```
% su
# mount /cdrom
# cd /cdrom/packages/ALL
```

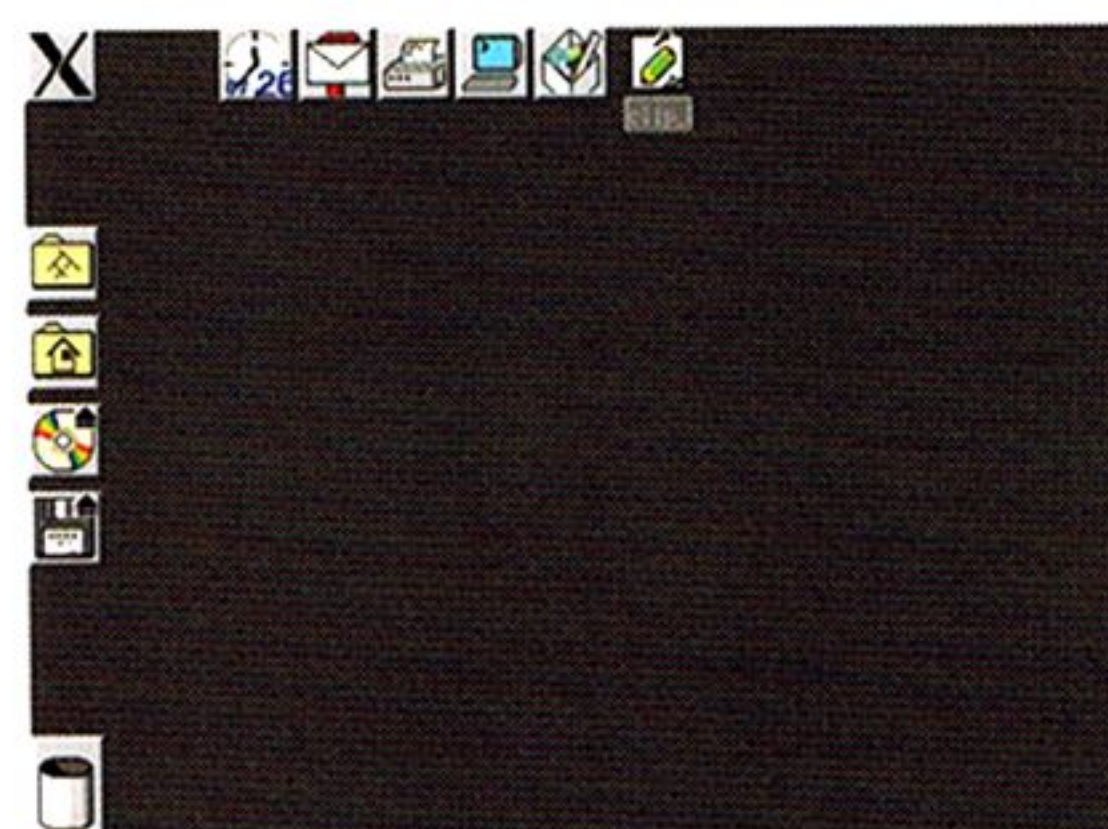



図4

```
# pkg_add fvwm95-2.0.43a.tgz
# pkg_add ImageMagick-4.0.7.tgz
# pkg_add ja-tcl-8.0.2.tgz
# pkg_add ja-tk-8.0.2.tgz
```

ImageMagick-4.1.4との相性が悪いことが確認されています。FreeBSD-2.2.8で実行する人は注意してください（アイコンの表示がおかしくなります）。

専用ユーザーの作成

デフォルトのインストーラは $HOME 配下のユーザーファイルを書き換えにいきます。現在の環境を破壊しないようにするため新しいユーザーを作成してそのユーザーでログインし直してください。

TkDesktopのインストール

TkDesktopの圧縮ファイルを展開しインストールを行います。

```
% tar -xvzf TkDesktop.tgz
% cd TkDesktop
% make install
```

いったんログアウトしxdmからログインし直せば自動的に起動します。

終わりに

このプログラムの仕様は私の趣味に偏ったものになっており，使う人によってさまざまな不満が出てくると思います。そういった方々はぜひ，このプログラムを拡張/変更してください。

リスト1

```
PixmapPath $HOME/.TkDesktop/images/:/usr/X11R6/icons/
IconPath   $HOME/.TkDesktop/images/:/usr/X11R6/icons/

Style "*" IconBox 305 0 900 100
Style "*" BorderWidth 3, HandleWidth 3
Style "*" Icon xlogo.xbm,  TitleIcon mini-x2.xpm

Style "NoTitle"    NoTitle, NoHandles, Sticky,BorderWidth 0, HandleWidth 0,WindowListSkip
Style "xicons*"    NoTitle, NoHandles, Sticky,BorderWidth 0, HandleWidth 0,WindowListSkip
Style "icon-*"     NoTitle, NoHandles, Sticky,BorderWidth 0, HandleWidth 0,WindowListSkip
Style "xtrash"     NoTitle, NoHandles, Sticky,BorderWidth 0, HandleWidth 0,WindowListSkip
Style "xconsole"   NoTitle, NoHandles, BorderWidth 3, HandleWidth 3
Style "kterm*"     Icon kterm.xpm,TitleIcon mini-term.xpm
Style "xdir*"      TitleIcon mini-xdir.xpm,Icon f_dir.xpm
Style "xed*"       TitleIcon mini-xed.xpm,Icon xed.xpm
Style "clip_board" Icon clip_board.xpm
```

mountsvr

1. 概要

mountsvr/mountcliは一般ユーザーからデバイスをマウント/アンマウントするためのツールです。

mountsvrはデーモンでスーパーユーザーから起動します。

mountcliはソケットを使用しmountsvrにmount/umountコマンドの実行を依頼します。

1.1 mountsvrの書式

```
mountsvr  [-l <path>][-s{ilole} <path>][-m <mode>][-g
<gid>][-d]
```

-lオプションはログを出力するファイルを指定します。指定がなければログは出力されません。

-sオプションはファイル名を指定します。2文字目がそれぞれ入力/出力/エラー出力を意味します。既定値は /tmp/.mountsvr {1|2|3} になります。

-mオプションはソケットのアクセスモードを8進数で指定します。既定値は，777です。

-gオプションはソケットのグループを番号で指定します。既定値は，0です。

-dオプションはデバッグ用でデーモンとして起動しません。

1.2 mountcliの書式

```
mountcli {+|-} <node>
```

第1引数が + = mount，- = umountを意味します。<node>は/etc/fstabに定義されているパス名です。

1.3 ejectの書式

```
eject <device>
```

未完成。

<device> で指定したデバイスがioctl（CDIOCEJECT）に対応していればメディアをイジェクトします。スーパーユーザーで実行する必要があります。

2. 機能

mountsvrはmountcliから要求を受け取りmount/umountを起動します。mount/umountの出力はソケットを通じてmountcliに送られ，mountcliの結果として出力されます。

マウントの要求は/etc/fstabにnoautoオプション付きで定義されているデバイスに限り，それ以外はエラーとします。

3. セキュリティ

mountsvr のセキュリティはソケットファイルへのアクセス制限によって行います。具体的にはアクセスモードを770に設定し，特定のグループに属するユーザーのみが実行可能なようにします。

4. インストール

インストールは以下の手順で行ってください。

```
> make all
> su
# make install
/etc/rc.localに以下の内容を追加します。
# mount server
  if [ -f /usr/local/bin/mountsvr ]; then
        echo -n ' mountsvr'
        /usr/local/bin/mountsvr -l /var/log/mountsvr -m
770 -g 100
  fi
```

xdirの概要

xdirはユーザーがカスタマイズ可能なファイラーです。

1.1 書式

```
xdir [ <directry> ]
```

<directry> は対象となるディレクトリです。省略時はカレントです。

1.2 一般的な操作法

xdir は一般的に次のような操作体系を持ちます。

左クリック：選択
左ダブルクリック：実行
右ドラッグ：メニュー
左ドラッグ＆ドロップ：コピー
左ドラッグ＆ドロップ+CTRL：移動
左ドラッグ＆ドロップ+SHIFT：リンク

2. ファイルアイコン

ファイルアイコンはシンボルとなるイメージとファイル名からなります。

アイコンはファイルの属性，名前によって異なりユーザー定義です。「初期化ファイル」の項を参照してください。ファイルの各モードは以下のように表現されます。

読み込み禁止：アイコン全体にメッシュ
書き込み禁止：右上に赤い禁止マーク
実行可能：左上に青い感嘆符付き
リンク ：右下に緑の矢印

2.1 ファイルアイコンの選択

2.1.1 ひとつのファイルを選択

ファイルアイコンを左クリックするとアイコンの周囲がへこんだようになります。この状態が選択された状態です。このとき，セレクションには選択されたファイルのフルパス名が設定されます。

2.1.2 複数のファイルを選択

ファイルアイコンのない位置から左ドラッグするとカーソルの位置に従って矩形が現れます。ドラッグを終了するとその時点で矩形内に含まれていたすべてのファイルアイコンが選択されます。

セレクションには選択されたすべてのファイルのフルパス名を空白で区切り設定します。

2.2 ファイルアイコンの実行

2.2.1 標準処理の実行

ファイルアイコンを左ダブルクリックするとそのアイコンに定義されている標準の処理を実行します。標準処理はユーザー定義です。「初期化ファイル」の項を参照してください。

2.2.2 メニューからの実行

ファイルアイコンを右ドラッグするとそのアイコンに定義されているメニューが出てきます。そのメニューの中から必要な処理を実行することができます。

メニューはユーザー定義です。「初期化ファイル」の項を参照してください。

2.3 ファイルアイコンによるファイル操作

2.3.1 ファイルの複写

まず，複写元のファイルのあるディレクトリと複写先のディレクトリを開きます。複写元ファイルをドラッグし複写先ディレクトリでドロップします。

または，複写元ファイルを選択し複写先ディレクトリ上で中クリックします。

2.3.2 ファイルの移動

まず，移動元のファイルのあるディレクトリと移動先のディレクトリを開きます。複写元ファイルをCTRL＋ドラッグし複写先ディレクトリでドロップします。または，移動元ファイルを選択し移動先ディレクトリ上でCTRL＋中クリックします。

2.3.3 ファイルのリンク

まず，リンク元のファイルのあるディレクトリとリンク先のディレクトリを開きます。

複写元ファイルをSHIFT＋ドラッグし複写先ディレクトリでドロップします。または，リンク元ファイルを選択しリンク先ディレクトリ上でSHIFT＋中クリックします。

2.3.4 ファイルの削除

削除ファイルをドラッグしxtrashアイコン上でドロップします。または，削除ファイルを選択しxtrashアイコン上で中クリックします。

3. ボタン

3.1 プッシュピンボタン

左クリックするとプッシュピンが刺さった状態と抜かれた状態に交互に切替えます。

プッシュピンが抜けていれば，ディレクトリ移動時に元のディレクトリウィンドウは消えてしまいます。

プッシュピンが刺さっていれば，ディレクトリ移動時に元のディレクトリウィンドウを残したまま新しいウインドウが表示されます。

3.2 ソートボタン

左クリックするとファイルアイコンを整列させます。右ドラッグするとソートの種別を選択するメニューが出ます。

3.3 フラッシュボタン

左クリックすると実際のディレクトリの内容と整合性を取ろうとします。コマンドラインからのファイル操作によって整合性が崩れたときに使用します。ソートボタンは同様の機能を含みます。

3.4 セーブボタン

左クリックすると現在の環境をファイルに保存し，次に起動されたとき再現できるようにします。

右ドラッグすると Auto と Manual 選択するメニューが出ます。Auto を選択すると終了時に自動的に環境を保存します。環境ファイルは˜/.TkDesktop/xdir.savedに保存されます。内容は初期化ファイルと同じです。

3.5 ビューボタン

左クリックしてもなにも起こりません。

右ドラッグするとアイコン表示に関するオプションメニューが出ます。

メニューの内容は，

Hidden file：ドットで始まるファイルの表示
Parent dir：親ディレクトリの表示
Executable mark：実行権マークの表示
Write protect mark：書き込み禁止マークの表示
Read protect mark：読み込み禁止マークの表示
Symbolic link mark：リンクマークの表示
Select icon frame：選択アイコンに枠を表示
Select icon negate：選択アイコンを反転表示
Image view：イメージファイルの場合，自分自身をアイコン化
Detail view：未実装

3.6 アップボタン

親ディレクトリに移動します。

4. 初期化ファイル

初期化ファイルはxdirが起動時に読み込むTcl/Tkスクリプトです。ファイル名は˜/.TkDesktop/xdir.defaultです。アイコンとメニューの定義についてはマクロが用意され簡単に記述できるようになっています。

4.1 アイコン定義

ファイルアイコンは以下のように定義します。

ICON ＜マスク＞ ＜イメージ＞ [＜オプション＞...]
＜オプション＞
-type ＜ファイル属性＞
-command ＜コマンド＞
-menu ＜メニュー＞

マスク：ファイル名とマッチさせる正規表現
イメージ：アイコンのイメージ・ファイル名。Bitmap（*.xbm），GIF（*.gif），PPM（*.ppm）のフォーマットが使用可能
ファイル属性：exec，dir，otherのいずれか。既定値はother
コマンド：ダブルクリック時の処理。Tcl/Tk スクリプト
メニュー：ユーザ定義のメニュー名

ファイルが複数の定義とマッチする場合，先に定義されているものが優先されます。

4.2 メニュー定義

メニューは以下のように定義します。

```
MENU <メニュー名> {
<項目名> <処理>
        :
        :
}
```

メニュー名 ：メニューの名前
項目名 ：メニュー項目名
処理 ：処理。Tcl/Tk スクリプト

メニュー名がwindow_menuのメニューはアイコンのない位置で右ドラッグしたときに出すメニューで必須です。

項目名に "----" を指定するとセパレータになります。

4.3 グローバル変数

SEL_NAME：選択アイコン名
PIN ：プッシュピンの状態 1 or 0
SORT ：ソートの種別 "Attribute" or "Name" or "Time" or "Wild Card"
SAVE ：環境の保存 "Auto" or "Manual"
WIN_W ：ウィンドウ幅
WIN_H ：ウィンドウ高
PAD_W ：アイコン間隔
PAD_H ：アイコン間隔
WMTYPE ：ウィンドウマネージャ "fvwm" or "twm"
SEND_DROP：ドロップイベントを送るプログラム名のリスト

4.4 プロシージャ

定義済みのプロシージャです。

Chdir ：ディレクトリの移動
Mkdir ：ディレクトリの作成
Rename ：ファイルのリネーム
Copy ：ファイルの複製を作る
One_Name {title init_text geometry action}
：1行入力

xedの概要

xed は xdir をサポートするための簡易エディタです。基本的にはテキストビューアとして利用されます。

1.1 書式

xed [-l] [-S] [<file name>]

<file name> は編集対象となるファイルです。-lオプションは編集中のファイルに対してほかのプロセスが変更を行うとそれを反映します。tail コマンドに似た動作をします。-Sオプションは入力ファイルを標準入力から取ります。

2 編集機能

xed は以下の編集機能を持ちます。

・text ウィジェット標準の編集機能
・検索と置換
・ほかのxedと共有するカットバッファ
・タグジャンプ

2.1 text ウィジェット標準の編集機能

textウィジェットのマニュアルを参照。

2.2 検索と置換

虫眼鏡のボタンをクリックすると検索用のサブウィンドウが開きます。以下の設定項目があります。

Search：検索文字列
Replace：置換文字列
Direction：検索方向
Case：英大/小文字を異なるものとする
Regexp：正規表現の使用

以下のボタンがあります。

Search：検索を実行しカーソルを移動します。検索された部分はセレクションされます。
Replace：セレクションされている部分を置換文字列で置き換えます。
All Replace：カーソル位置からファイルの終わり（先頭）まですべて置き換えます。
Quit：ウィンドウを閉じます。

※Tk8.0jpでは検索機能に日本語が使えません。

2.3 ほかのxedと共有するカットバッファ

左ボタンのドラッグによりテキストをセレクションした状態で右ボタンによりメニューを開きます。メニューのCopy，Cut，Pasteを選択することによりそれぞれの機能が働きます。

Copy，Cut したテキストはほかのxedによって Pasteすることができます。この機能を使用する場合はclip_boardを常駐させておいてください。

また，CTRL＋ドラッグで矩形のカット＆ペーストができます。

2.4 タグジャンプ

タグ形式の行にカーソルを移動させ右ボタンのメニューからTagを選択します。

2.5 指定行へ移動

右上の数値は左から行番号とカラムを表しています。行番号を左クリックすると入力モードになり数値を入力し改行を入力すれば指定行へ移動します。

また，中クリックでセレクションのペースト，右クリックで改行キーと同じ機能があります。

3 保存と終了

3.1 保存

いちばん左のボタンをクリックすると保存のダイアログが表示されます。ダイアログには以下のボタンがあります。

Save：ファイルを保存し編集に戻ります
Save&Quit：ファイルを保存しxedを終了します
as Name：別名でファイルを保存します
GoBack：なにもせず編集に戻ります

3.2 終了

終了はウィンドウマネージャからWM_DELETE_WINDOWを発行して行います。ファイルが変更されていなければそのまま終了します。ファイルが変更されていた場合，ダイアログが表示されます。

Save：ファイルを保存し xed を終了します
GoBack：なにもせず編集に戻ります
Quit：ファイルを保存せずしxedを終了します

4. 初期化ファイル

初期化ファイルはxedが起動時に読み込むTcl/Tkスクリプトです。ファイル名は˜/.TkDesktop/xed.default です。ユーザー定義のキーバインドなどを定義します。

xiconsの概要

xicons はデスクトップアイコンです。

1.1 書式

xicons

1.2 一般的な操作法

xiconsは一般的に次のような操作体系を持ちます。

左ダブルクリック：実行
左ドラッグ＆ドロップ：ファイルのドロップ
中クリック：ファイルのドロップ
右クリックor ドラッグ：メニュー

2. 定義ファイル

定義ファイルはxiconsが起動時に読み込む定義ファイルです。ファイル名は˜/.TkDesktop/xicons.defaultです。

アイコンとメニューの定義についてはマクロが用意され簡単に記述できるようになっています。

2.1 アイコン定義

アイコンは以下のように定義します。

ICON ＜名前＞ ＜位置＞ ＜イメージ＞ [＜オプション＞...]
＜オプション＞
-command ＜コマンド＞
-menu ＜メニュー＞
-eject ＜ejectコマンド＞
-bitmap ＜ビットマップ＞
-text ＜テキスト＞
-drop ＜ドロップコマンド＞
-root

名前：アイコンの名前
位置：アイコンを置く位置。書式はジオメトリと同じ
イメージ：アイコンのイメージファイル名。Bitmap（*.xbm），GIF（*.gif），PPM（*.ppm）のフォーマットが使用可能。Tkが拡張されていればPixmap（*.xpm）も使用可能
コマンド：ダブルクリック時の処理。Tcl/Tkスクリプト
メニュー：ユーザー定義のメニュー名
ejectコマンド：ejectボタンを押されたときの処理。Tcl/Tk スクリプト
ビットマップ：イメージの上にかぶせるビットマップ
テキスト：イメージの上にかぶせるテキスト
ドロップコマンド：ドラッグ＆ドロップされたときの処理。Tcl/Tk スクリプト

-root オプションはXロゴウィンドウの代わりのアイコンを作成するときつけます。

2.2 メニュー定義

メニューは以下のように定義します。

```
MENU <メニュー名> {
  <項目名> <処理>
        :
        :
}
```

メニュー名 ：メニューの名前
項目名 ：メニュー項目名
処理 ：処理。Tcl/Tkスクリプト

Background Mascot Shell & 開発ツール

菊地 功　Kikuchi Isawo

Windowsの背景でさまざまなキャラクターが動き回る……それがBGマスコットシェルシステムだ。数年前から制作されていたものだが，今回初めて開発環境が公開される。X68000でお馴染みだったFishもデラックス版となってCD-ROMに収録されている。

その昔，Oh!Xの付録ディスクに収録された，X680x0シリーズで動作する熱帯魚の環境ソフト(？)のことを覚えているだろうか？

横内氏によって作成された，背景がゆらゆらとラスタスクロールし，色とりどりの熱帯魚が泳ぎ回るアレである。石上氏によってSX-Windowのスクリーンセーバーになったが，のちに私もWindowsへの移植を試みた。ただ，なぜか当時の私はスクリーンセーバーにしようとは考えなかった。かといって，ウィンドウが開いて，そのなかで魚が泳いでいるだけでは面白くない。その当時，「チャ○ンス」という，Windowsのデスクトップ(当時はまだWindows 3.1)で，とあるアニメのキャラクターが踊りまくるというアクセサリが流行っていた(68でもあったなぁ)。しかも，ありがたいことにソースまで無償配布されていたりした。「これだ！」ということで最初に作ったのが"Fish for Windows"である(図1)。要するに，デスクトップに熱帯魚を泳がせてしまおうというわけだ(背景のラスタスクロールはできないが)。

Fishはシェアウェアであり，一部の人の反感を買ったようだが，Oh!Xが休刊となり，当時すでにフリーのライターであった私はかなり財政事情が苦しかったのである。このFishは結構多くの人に支持されたのだが，なにせ単体で閉じてしまったシステムである。このままではこれ以上の発展がないと考えた私は，そののちにシステムとキャラクター(熱帯魚)を切り離すことを考えた。これがBackground Mascot ShellとFish for MascotShellである(図2)。

つまり，シェル側は描画などを担当，キャラクターはマスコットデータとして分離して，別途にデータを用意すればさまざまなキャラクターをデスクトップでアニメーションさせることができるわけだ。複数のデータをロードさせることができるので，別々に作られたデータを混在表示することもできる。

この性質を利用して，Fishの追加モジュール(図3)も作成したのだが，マスコットデータとはいえ，その実はDLL，つまりプログラムであるために，作成にはそれなりの開発環境を必要とし，しかもシェル自体がまだ未完成ということもあって，ライブラリなどを外に出せない状態であった。つまり，データが作成できるのは私だけだったのだ。

とりあえず画像データを作ってもらえば，私が代行してデータを制作するサービスを行っていたのだが，どうも思うようにデータが増えない。増えたら増えたで私が死んでしまう。かといって，せっかく確立したシェルがこのまま立ち腐れというのももったいない。

なにかしらの開発キットを作らなきゃなぁ，と思っていたところで問題が発生した。Internet Explorer 4.0のアクティブデスクトップと，Windows 98の登場である。双方とも，はた目にはあまり違いなさそうに見えるが，いままでとはデスクトップの扱いがちょっと違ってしまっている。幸いアクティブデスクトップは，私の撲滅キャンペーンのおかげ(じゃないだろうが)でInternet Explorer 5.0では排除されたが，Windows 98への対応でちょっと手間取ってしまった。アクティブデスクトップへの対応は完全ではないし，Windows 98での動作はまだちょっと怪しい。

もうちょっといえば，ついこの前と比べてかなり重くなったような気もする。ちょっと調べたところ，デスクトップの更新にかなりのウェイトが入っているようだ。原因はまだわかっていないが，DirectX 6.1か，その辺りがちょっと怪しい。ただ，完成度自体はそれなりに上がってきていると自分では思っている。そこで，門外不出であったライブラリ，および即席の開発キット"Mascot Development Kit"略してMDKを発表し，広く一般にデータの作成をお願いしたい。ただ，シェルの仕様変更の可能性もまだないわけではないのと，もろもろの事情により，ここで発表するものは転載・再配布禁止とする。なお，ここではシェル自体の使用法は説明しないので，まずそちらのヘルプを参照してほしい。

ネイティブライブラリ編

当初からデータの作成に用いられていた手法である。つまり，生成されるデータはDLL(拡張子はMSCとされている)であり，別途開発環境を必要とする。想定されているのはVisual C++ 6.0であり，スケルトンプログラムを用意してあるので，典型的なデータならばちょっとした変更で作成できる(サンプル・Skeleton)。このスケルトンは，コンパイルするとすでに動くものができるようになっている(図4)。ガイコツクラゲモドキが下から上へ向かって昇っていくだけだが，よく見ると腕(歯？)が水をかいて，その勢いで上がっていくかのように速度に揺らぎをつけている。マスコットデー

図1　Fish for Windows。Windows 3.1版と95版があった

図2　Background Mascot ShellとFish for MascotShell

図3　Fishの追加モジュール。Fishとともに当初はシェアウェアであったが，現在はフリー

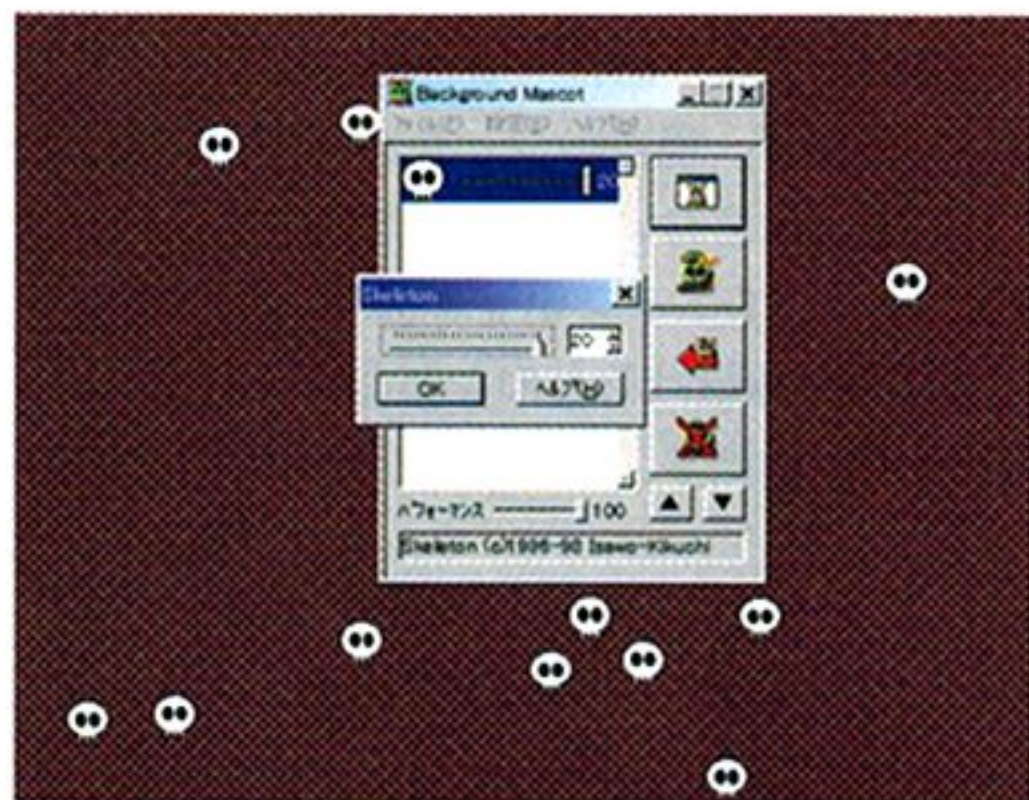



図4　シンプルで基本的なデータスケルトン

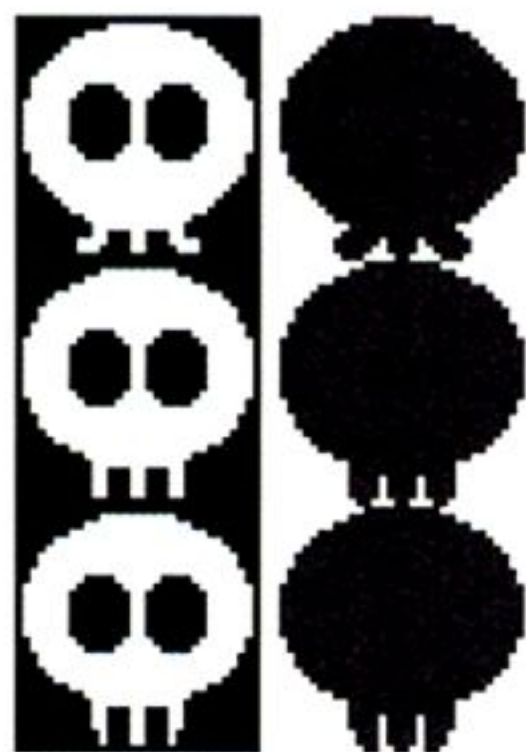



図5，6　キャラクターのもととなるカラーデータとマスクデータ

タの基本「アニメーション」と「座標の算出」をきっちり踏襲しているわけだ。

それらを制御するのは，主にMascot.cpp内のCMascotDll::SetPattern()メソッド内である（リスト1）。このforループのコメントを見れば，だいたいの感じは理解できるだろう。CMascotというクラスの配列に，キャラクターの座標その他もろもろを格納してやるだけでいいのだ。キャラクター座標は画面左上が原点で，右向きがX，左向きがYであり，キャラクター画像の左上がその座標となるように配置される。デバイスコンテキストのほうは，別のところでm_hMascotDCには図5，m_hMascotMDCには図6のような画像がすでに格納されている。

ここで，この2つの画像について説明しておこう。図5のほうは，単純にカラー画像である（このキャラクターはモノクロだが）。つまり，実際に画面に描画される画像となる。ただし，背景部分は黒にしておかなければならない。それに対し，図6は前景と背景部分を区別するマスク画像であり，前景，つまりキャラクター部分は黒で，背景部分は白で描画されたモノクロ画像としておく。CMascotDll::SetPattern()メソッドは約0.1秒間隔でシェルから呼び出され，それによってCMascotクラス配列に格納されたデータをもとにデスクトップに描画されるので，mascot[i].m_DCyとmascot[i].m_MDCyに対して，呼ばれるたびに0，CHARYSIZE，CHARYSIZE＊2，0，CHARYSIZE，……という値を入れてやれば，上から順にアニメーションすることになる（CHARYSIZEはキャラクターの高さ）。

このサンプルでは，CMascotDllクラスのメンバm_x[i]，m_y[i]にキャラクターの座標を保持してあり，m_numというのはスライダで指定されている最大表示数である。m_y[i]の減算量（つまり上向きの速度）がちょっと煩雑だが，これが速度の揺らぎとなっている。

また，その下のif文は画面上端に達したときの処理である。もちろん描画を停止させるのだが，そのm_NewShowにFALSEを入れるとともに，StackQueue()という関数が呼ばれている。これはFIFOバッファであるキューにキャラクターナンバーを放り込む操作である。

すぐに次を表示せずに，わざわざこうしているのには理由がある。シェルにはパフォーマンススライダというものがあり，これを使ってシェルが消費するマシンパワーを抑制することを目指している。シェル側はまだマシンに余力があると，CMascotDll::SetPattern()の第2引数EntryにTRUEを入れてよこすが，そうでないときはFALSEとなる。つまり，FALSEは，

「これ以上負荷を増やしてはいけないよ」

というシェルからのメッセージなのである。したがって，データ側は表示しているキャラクターがまだ設定されたキャラクター数に達していなくても，それ以上キャラクターを出現させてはいけない。このために待ち行列を作って，EntryにTRUEが入ったときだけキャラクターを投入するようにしているのだ。

もうひとつ重要なのが，CMascotDll::InitChar()メソッドだ。動きを設定しても，初期位置がわからなければ登場できない。このサンプルでは，画面の下からの登場しかないので，X座標には画面全域をカバーするランダム値（m_DesktopSizeはデスクトップサイズが入っている），Y座標には画面の下端が入れてある。これが初歩の初歩である。

シーラカンス

では，このスケルトンから新しいマスコットデータを作ってみよう。題材としては，Direct3D Retained Modeでやったシーラカンスを使い回してみる。まずは，そちらから図7および図8のような画像データを作成する。ぱっと見に同じものが並んでいるだけのように見えるかもしれないが，ちゃんとアニメーションするデータになっている。また，右向き画像は左向きを反転させただけだが，シェルには画像の反転描画機能はないので，データ側で持たせておく必要がある。カラーデータは，多色で持たせるとサイズが大きくなるので，16色ないしは256色のRLE圧縮したBMPで作成することをおすすめする。さて，これをresフォルダの中にそれぞれchar.bmpとmask.bmpという名で上書きしてしまおう。

そうしたら，まずは細かい部分の修正だ。アイコンやダイアログ，バージョンリソースを適当に書き直す（図9）。このとき，そのデータがシェルのど

リスト1　SetPatternメソッドのスケルトン

```
int CMascotDll::SetPattern( CMascot *mascot, BOOL Entry, UINT Count )
{
   // MascotShellがパターンを取得するときに呼ばれる関数
   // Count は，呼ばれるたびにインクリメントされる，すべてのマスコットデータに
   // 共通のカウンタ
   int i;
   if( Entry ){    // Entry が真のときだけキャラクタの新規投入を許す
       // キューからキャラクタを取得
       if( FirstEntry!=LastEntry ){
           i = Queue[FirstEntry++];
           IsQueue[i] = FALSE;
           if( FirstEntry>MAXCHARNUM ) FirstEntry = 0;
           if( i<m_num ){
               if( !mascot[i].m_NewShow ){
                   mascot[i].m_NewShow = TRUE;
                   InitChar( i );
               }
           }
       }
   }
   for( i=0; i<MAXCHARNUM; i++ ){
       if( i<m_num && mascot[i].m_NewShow ){
           // キャラクタの動作を記述
           // キャラクタが画面外に出た場合などは
           // StackQueue( i );
           // mascot[i].m_NewShow = FALSE;
           // として，キャラクタ表示をキャンセルさせる。
           // mascot[i].m_Newx = 画面上でのキャラクタX座標;
           // mascot[i].m_Newy = 画面上でのキャラクタY座標;
           // mascot[i].m_NewSize.cx = キャラクタXサイズ;
           // mascot[i].m_NewSize.cy = キャラクタYサイズ;
           // mascot[i].m_hDC = キャラクタのデバイスコンテキストm_hMascotDC;
           // mascot[i].m_hMDC = マスクのデバイスコンテキストm_hMascotMDC;
           // mascot[i].m_DCx = m_hMascotDC内でのキャラクタX座標
           // mascot[i].m_DCy = m_hMascotDC内でのキャラクタY座標
           // mascot[i].m_MDCx = m_hMascotMDC内でのキャラクタX座標
           // mascot[i].m_MDCy = m_hMascotMDC内でのキャラクタY座標
           // 以下サンプル
           if( ++m_count[i]>=CHARPATTERN ) m_count[i] = 0;
           mascot[i].m_Newx = m_x[i];
           mascot[i].m_Newy = m_y[i]-=(m_count[i]<<3)+4+(rand()>>13);
           if( mascot[i].m_Newy<=-32 ){
               StackQueue( i );
               mascot[i].m_NewShow = FALSE;
               continue;
           }
           mascot[i].m_NewSize.cx = CHARXSIZE;
           mascot[i].m_NewSize.cy = CHARYSIZE;
           mascot[i].m_hDC = m_hMascotDC;
           mascot[i].m_hMDC = m_hMascotMDC;
           mascot[i].m_DCx = mascot[i].m_MDCx = 0;
           mascot[i].m_DCy = mascot[i].m_MDCy = m_count[i]<<5;
           // ここまでサンプル
       } else mascot[i].m_NewShow = FALSE;
   }
   return m_num;
}

void CMascotDll::InitChar( int n )
{
   // キャラクタがキューから取り出された後の初期化処理
   // 以下サンプル
   m_x[n] = (rand()*(m_DesktopSize.cx-32))>>15;
   m_y[n] = m_DesktopSize.cy;
   m_count[n] = 0;
   // ここまでサンプル
}
```

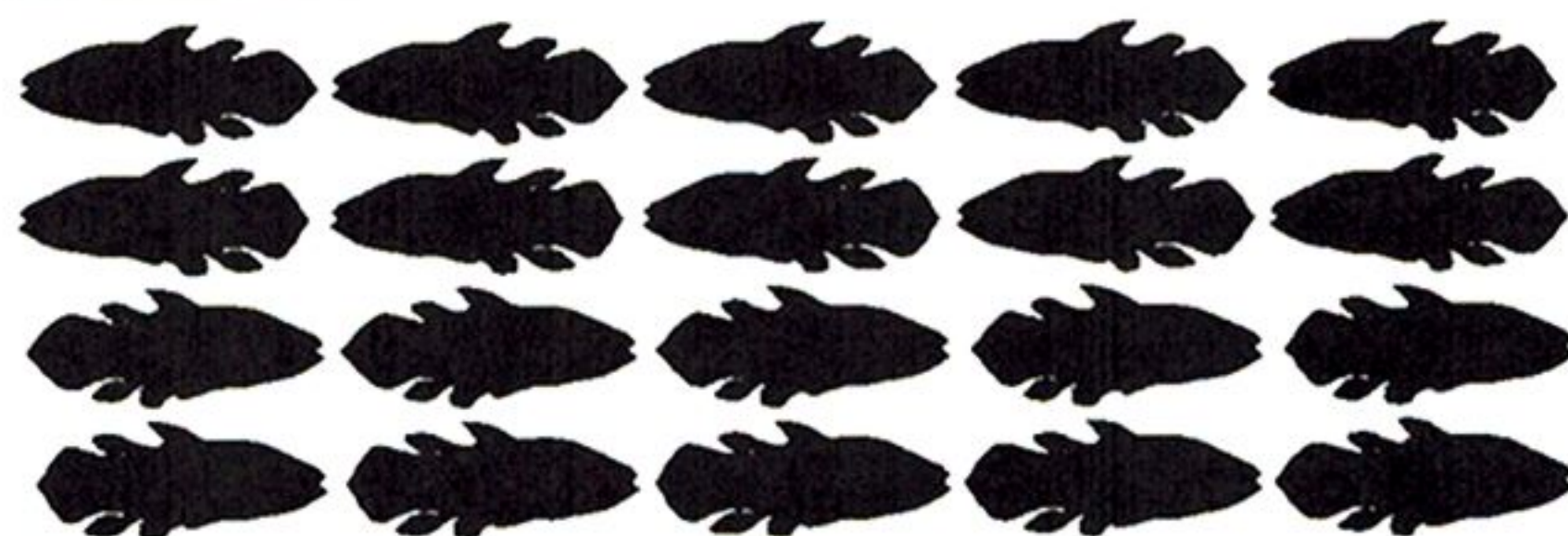
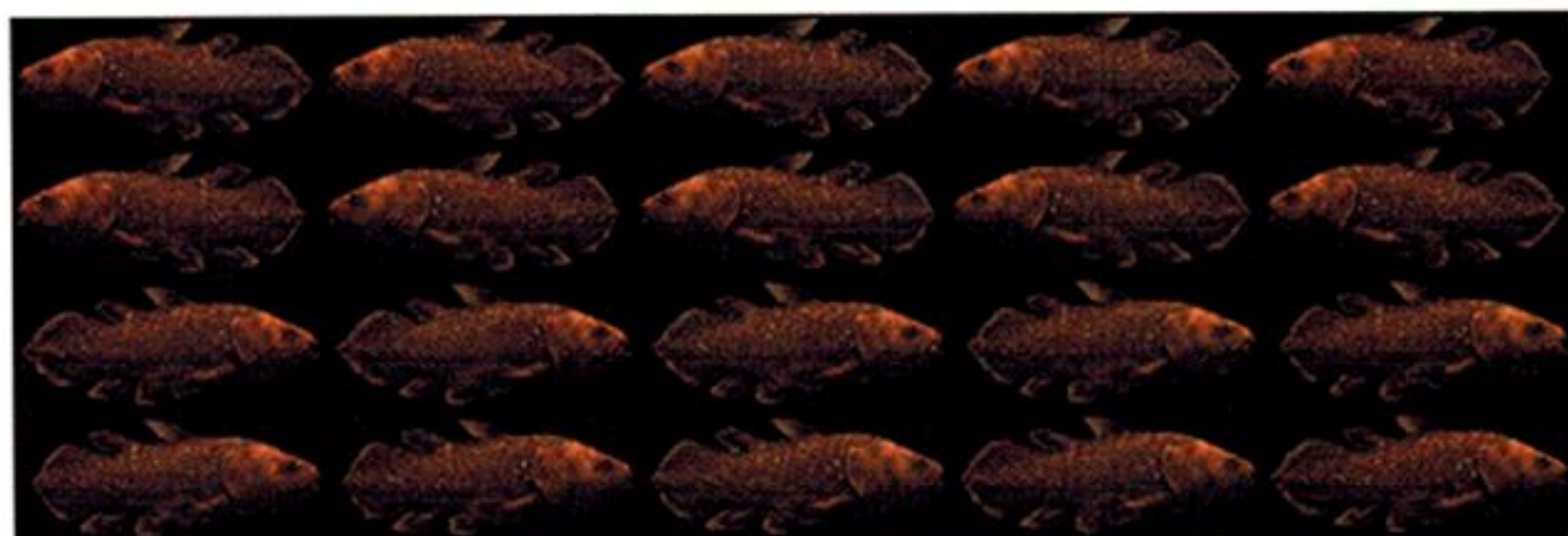



図7，8　シーラカンスのカラーデータとマスクデータ

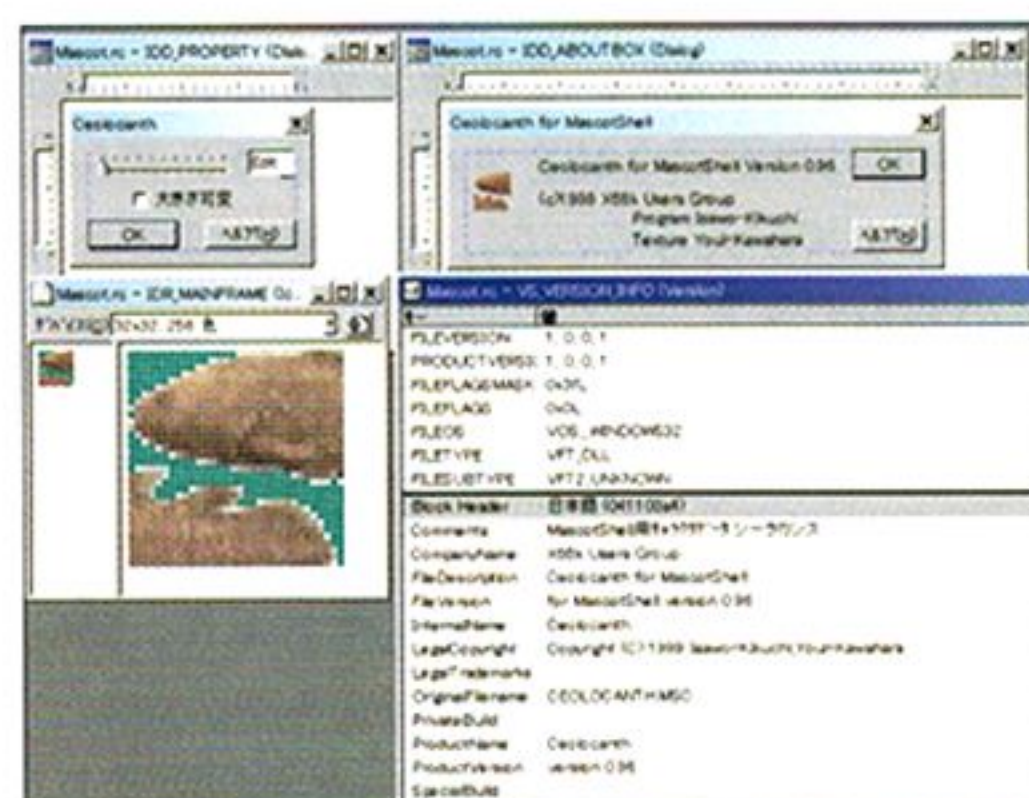



図9　各種リソースの書き換え

のバージョンに対応しているかを明確にするために，データバージョンをシェルバージョンにあわせることにしている。現在のシェルバージョンは0.96であるので，データバージョンも0.96にしておいてもらいたい。また，プロパティダイアログに「大きさ可変」というチェックボックスをつけたことに注意だ。だいたい想像はつくだろうが，詳細はあとのお楽しみだ。

それが済んだら，今度はMascot.cppを開き，先頭のComment()という関数の，

```
if( pShellInfo->nVersion<0 * 256+96 ){
  MessageBox( pShellInfo->hWnd, "このデータは
  MascotShell ver0.96以上が必要です。",
  "Skeleton for MascotShell", MB_OKIMB_ICONHAND );
  return NULL;
}
```

という部分も，

```
if( pShellInfo->nVersion<0 * 256+96 ){
  MessageBox( pShellInfo->hWnd, "このデータは
  MascotShell ver0.96以上が必要です。",
  "Ceolocanth for MascotShell", MB_OKIMB_ICONHAND );
  return NULL;
}
```

と書き直しておく。pShellInfo->nVersionというのはシェルから渡されるバージョンであり，もしそれが0.96よりも以前のバージョンであれば，警告メッセージを出し，終了する。さらに，その関数のいちばん最後，

```
return "Skeleton (c)1996-99 Isawo-Kikuchi";
```

を，

```
return "Ceolocanth (c)1999 Isawo-K, Youi-K";
```

のように変更する。この文字列は，シェルのメインウィンドウのいちばん下

リスト2　処理の指定例

```
int CMascotDll::SetPattern( CMascot *mascot, BOOL Entry, UINT Count )
{
   // MascotShellがパターンを取得するときに呼ばれる関数
   // Count は，呼ばれる度にインクリメントされる，全てのマスコットデータに
   // 共通のカウンタ
   int i;
   if( Entry ){    // Entry が真のときだけキャラクタの新規投入を許す
       // キューからキャラクタを取得
       if( FirstEntry!=LastEntry ){
           i = Queue[FirstEntry++];
           IsQueue[i] = FALSE;
           if( FirstEntry>MAXCHARNUM ) FirstEntry = 0;
           if( i<m_num ){
               if( !mascot[i].m_NewShow ){
                   mascot[i].m_NewShow = TRUE;
                   InitChar( i );
                   mascot[i].m_bStretch = m_stretch;
                   mascot[i].m_NewDestSize = m_size[i];
               }
           }
       }
   }
   for( i=0; i<MAXCHARNUM; i++ ){
       if( i<m_num && mascot[i].m_NewShow ){
           // キャラクタの動作を記述
           if( ++m_count[i]>=CHARPATTERN ) m_count[i] = 0;
           m_x[i] += m_dx[i];
           m_dx[i] += ((rand()*3)>>15)-1;        // -1～1の加減速
           if( m_direct[i]==0 ){  // 左向き
               if( m_x[i]<-m_size[i].cx ){        // 画面外判定
                   StackQueue( i );
                   mascot[i].m_NewShow = FALSE;
                   continue;
               }
               // X速度はm_minspeed～m_maxspeed
               if( m_dx[i]<-m_maxspeed[i] ) m_dx[i] = -m_maxspeed[i];
               else if( m_dx[i]>-m_minspeed[i] ) m_dx[i] = -m_minspeed[i];
           } else {    // 右向き
               if( m_x[i]>=m_DesktopSize.cx ){  // 画面外判定
                   StackQueue( i );
                   mascot[i].m_NewShow = FALSE;
                   continue;
               }
               // X速度はm_minspeed～m_maxspeed
               if( m_dx[i]<m_minspeed[i] ) m_dx[i] = m_minspeed[i];
               else if( m_dx[i]>m_maxspeed[i] ) m_dx[i] = m_maxspeed[i];
           }
           m_y[i] += m_dy[i];
           m_dy[i] += ((rand()*3)>>15)-1;        // -1～1の加減速
           // Y速度は-1～1
           if( m_dy[i]<-1 ) m_dy[i] = -1;
           else if( m_dy[i]>1 ) m_dy[i] = 1;
           if( m_y[i]<-m_size[i].cy || m_y[i]>=m_DesktopSize.cy ){       // 画面外
判定
               StackQueue( i );
               mascot[i].m_NewShow = FALSE;
               continue;
           }
           mascot[i].m_Newx = m_x[i];
           mascot[i].m_Newy = m_y[i];
           mascot[i].m_NewSize.cx = CHARXSIZE;
           mascot[i].m_NewSize.cy = CHARYSIZE;
           mascot[i].m_hDC = m_hMascotDC;
           mascot[i].m_hMDC = m_hMascotMDC;
           mascot[i].m_DCx = mascot[i].m_MDCx = m_count[i]%5*CHARXSIZE;
           mascot[i].m_DCy = mascot[i].m_MDCy =
(m_count[i]/5+m_direct[i]*2)*CHARYSIZE;
       } else mascot[i].m_NewShow = FALSE;
   }
   return m_num;
}

void CMascotDll::InitChar( int n )
{
   // キャラクタがキューから取り出された後の初期化処理
   if( m_stretch ){
       int r = rand();
       // 本来の大きさの1/2～2倍
       m_size[n].cx = ((r*3*CHARXSIZE/2)>>15)+CHARXSIZE/2;
       m_size[n].cy = ((r*3*CHARYSIZE/2)>>15)+CHARYSIZE/2;
       m_minspeed[n] = (2*m_size[n].cx+CHARXSIZE/2)/CHARXSIZE;
       m_maxspeed[n] = (6*m_size[n].cx+CHARXSIZE/2)/CHARXSIZE;
   } else {
       m_size[n].cx = CHARXSIZE;
       m_size[n].cy = CHARYSIZE;
       m_minspeed[n] = 2;
       m_maxspeed[n] = 6;
   }
   m_direct[n] = (rand()*2)>>15;  // 0:左向き 1:右向き
   m_x[n] = m_direct[n]?-m_size[n].cx:m_DesktopSize.cx;
   m_y[n] = (rand()*(m_DesktopSize.cy-m_size[n].cy))>>15;
   m_dx[n] = ((rand()*5)>>15)+2;
   if( m_direct[n]==0 ) m_dx[n] = -m_dx[n];
   m_dy[n] = ((rand()*3)>>15)-1;
   m_count[n] = 0;
}
```

のコメント欄に表示されるメッセージである。それから，Mascot.hの中の文字定数を，

```
#define MAXCHARNUM    20    // 最大キャラクター数
#define DEFCHARNUM     4    // デフォルトキャラクター数
#define CHARPATTERN   10    // アニメーションパターン数
#define CHARXSIZE    190    // キャラクターXサイズ
#define CHARYSIZE     80    // キャラクターYサイズ
```

とでもしておこう。特に説明はいらないだろう。

さて，下準備はできた。動きの記述に入ろう。シーラカンスは左右から出現し，横方向に進むものとする。そのため，まずどちら向きなのかを保持するm_direct[]をCMascotDllのメンバに加え，これが0のときは左向き，1のときは右向きとする。ただし，真横ではなくて，適当に上下にも揺らしたほうが自然だろう。

また，移動量（＝速度）をランダムで決めてしまうと，不自然な動きになるので，速度m_dx[]，m_dy[]もメンバに加え，その速度に対してランダム値を加減算することにより，移動量にばらつきを与える。

それともうひとつ，先ほどの「大きさ可変」である。シェルには，拡大縮小の機能が備わっている。これを利用して，いろいろな大きさのシーラカンスを出現させようというわけだ。ただし，拡大縮小はそれなりに重いので，マシンによってはこれを使うと表示できるキャラクター数が極端に減ってしまうことがある。よって，このサンプルのように，拡大縮小はユーザー側が設定した場合のみ使うようにするべきだろう。拡大縮小を行うには，CMascotクラスのm_bStretchをTRUEにし，m_NewDestSizeに書き込まれ側のサイズを指定する。そのため，CMascotDllクラスに描画サイズm_size[]を追加している。また，サイズによって速度の範囲も変える必要があるので，最低速度m_minspeed[]と最大速度m_maxspeed[]もメンバに加える。

このようにしてできたのがリスト2だ。先ほどよりは複雑だが，基本部分はなにも変わっていないので，難しい部分は特にないだろう。m_stretchはプロパティダイアログのチェックボックスがチェックされているときはTRUE，そうでないときはFALSEとなる。もちろんプロパティダイアログもそのチェックボックス関係の部分を修正する必要があるが，ここはVisual C++の解説記事ではないので省略する。

図10　大小のシーラカンスがデスクトップを横切る

また，ヘルプファイルを作っておけば，ダイアログの［ヘルプ］ボタンを押したときにそのヘルプが表示されるようにもなっている。ヘルプファイルについては，EXプラグインのほうの記事を参照のこと。

こうしてできたマスコットデータが図10である（サンプル・Ceolocanth）。この程度のものならば，なんということもなく作れてしまうことがおわかりいただけただろう。実際問題として，このバリエーションを利用すれば，一般的なものはたいていカバーできてしまうだろう。しかし，まだまだこのくらいは序の口である。

メタルドラゴンbyどんふり

結局，動き自体はプログラムでいくらでも制御できるし，キャラクター同士の連携もプログラム次第である。そんなわけで，多関節モノなどもプログラムのウデ次第なわけだ。今回どんふり氏から提供していただいたデータ「メタルドラゴン」がそれに当たる（図11）。

頭ひとつと複数の胴体パーツがつながって，1匹のドラゴンをなす。もとになる画像データは図12，13だ。24方向の頭と胴体が並んでいる。ここでよく見てもらいたいのが影の部分だ。半透明のテクニックその1，マスクをメッシュにする。力を入れて説明するほどでもないが，それなりの解像度なら，これで背景が透けているように見える。

さて，ここで気づいてもらいたいのは，24方向への運動は，X，Yでは制御しにくいということだ。こういった場合は極座標を用いるのが正解だろう。角度と速度で制御し，それをX，Yに落とし込む。そのために角度m_direct[]とm_speed[]を新たにメンバに加え，次のようにX，Yに加減算する。

```
m_x[i] -= (int)(m_speed[i] * sin(m_direct[i] * 3.1415926/12));
m_y[i] += (int)(m_speed[i] * cos(m_direct[i] * 3.1415926/12) * 3/4);
```

m_direct[]は下向きが0（だってデータが下向きが最初だから），時計回りが正方向で，24が2πである。Yの最後に3/4をかけているのは，俯瞰図であるために，上向き（奥行き）に対してスケールダウンするためだ。ただし，ここでm_speed[]をドット単位にすると，演算による切り捨て誤差が大きくなりすぎる。そこで，m_speed[]およびm_x[]，m_y[]は8ビットの下駄をはかせて計算し，最後にCMascotのインスタンスに代入するときに8ビットシフトダウンするようにする。こうしておけば，小数部の誤差は蓄積され，微妙な角度もしっかり表現できる。

ところで図12のドラゴンは白（シルバー）だけだが，図11の予想図では青いドラゴンもいる。では青い画像も必要なのかというと，そんなことはない。じつはシェルはパレットアニメーションをサポートしているのだ。今回はキャラクター自体がパレットアニメーションするわけではないが，要するにパレットを用意すれば，画像を別途準備することなしに「色違い」キャラクターを表示できる。今回表示するのは，シルバーに加え，ブラック，レッド，グリーン，ブルーの計5色である（図14）。そこで，まずはこれらのパレットを作っておく。パレットはMicrosoftパレット形式で，私はPaint Shop Proで作っている。このパレットを保存するためのメンバm_PalNo[5]と

図11　多関節キャラクター「メタルドラゴン」の完成予想図

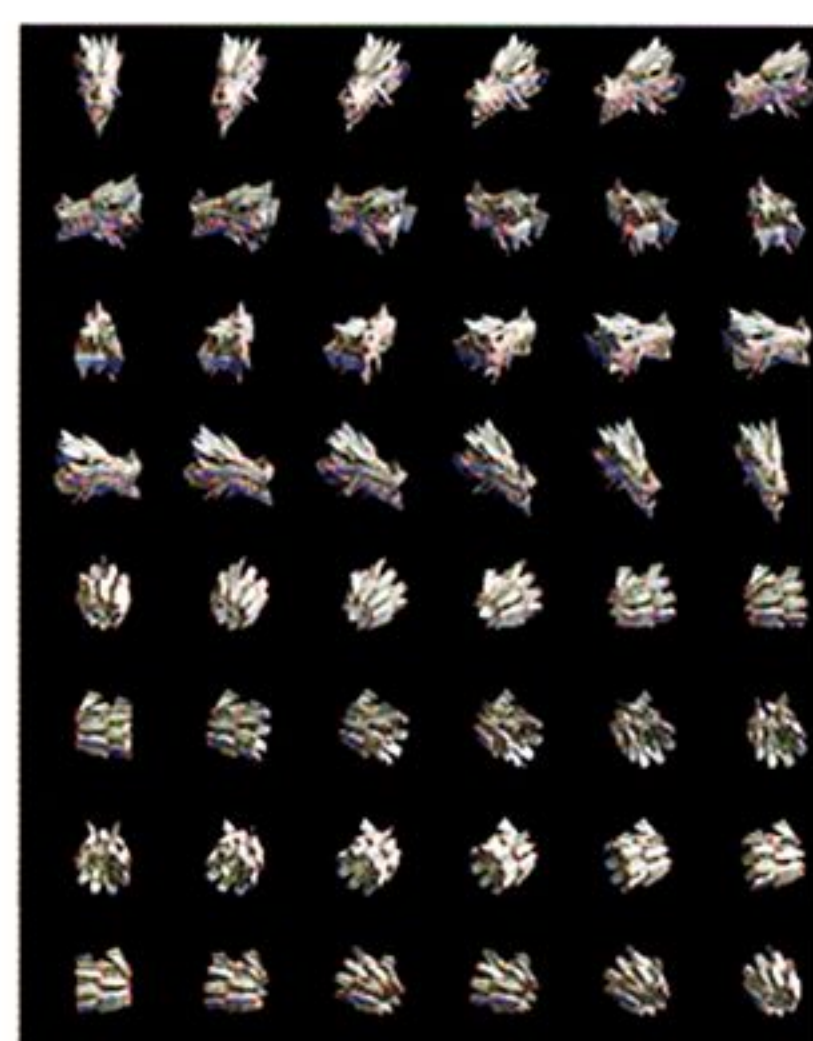



図12，13　メタルドラゴンのパーツ。なかなかの大作だ

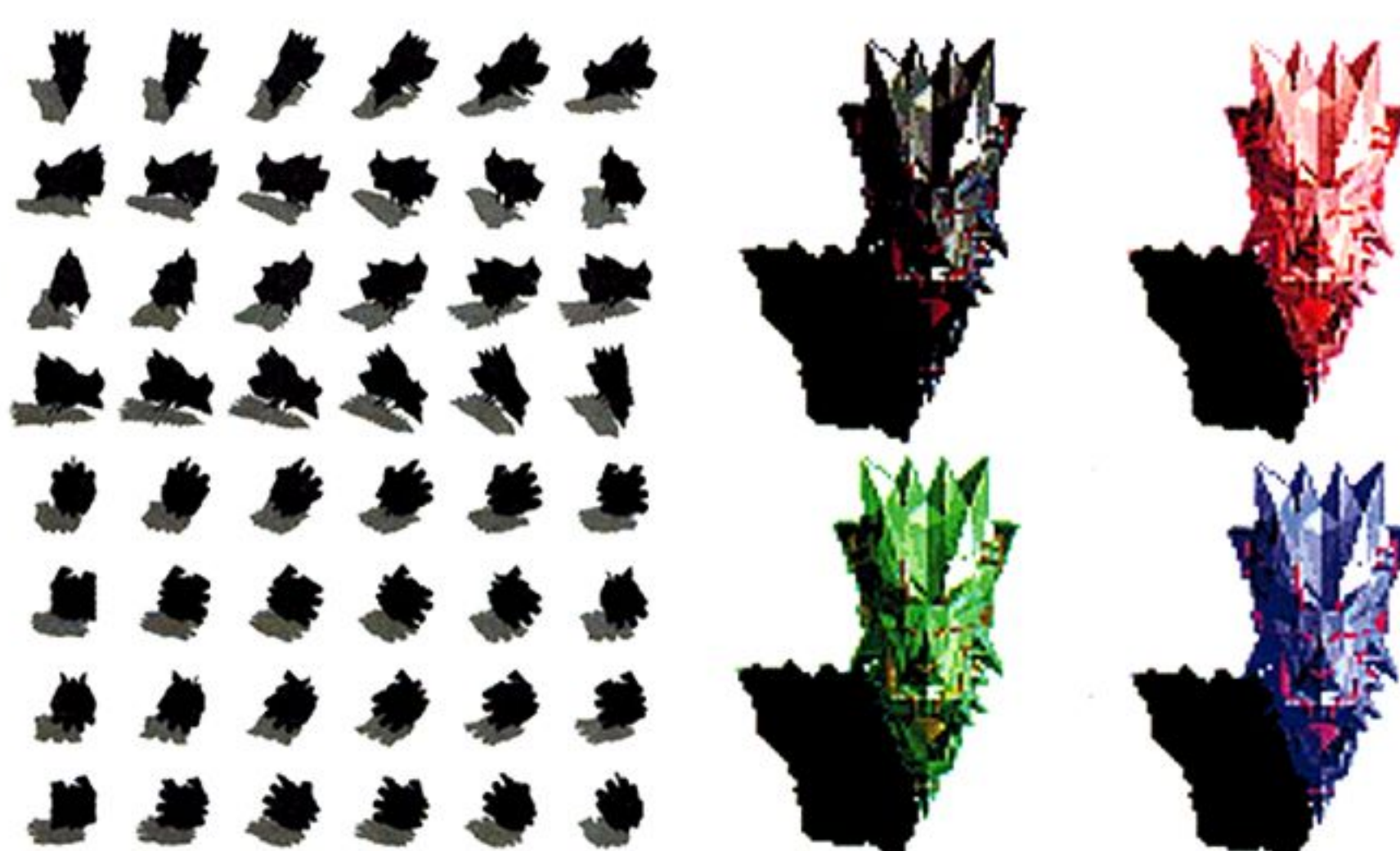



図14　ドラゴンのカラーバリエーション

RGBQUAD構造体m_pPalette[5]を追加しておいて，CMascotDll::LoadChar()内でカラー画像のロード部分を次のようにする。

```
//m_hMascotBitmap = LoadBitmap( hInstance,
                        MAKEINTRESOURCE(IDB_CHAR) );
m_PalNo[0] = LoadDIBSection( hInstance,
MAKEINTRESOURCE(IDB_CHAR), &m_hMascotBitmap,
                                    &m_pPalette[0] );
m_PalNo[1] = LoadPalette( hInstance,
     MAKEINTRESOURCE(IDR_BLACK), &m_pPalette[1] );
m_PalNo[2] = LoadPalette( hInstance,
         MAKEINTRESOURCE(IDR_RED), &m_pPalette[2] );
m_PalNo[3] = LoadPalette( hInstance,
     MAKEINTRESOURCE(IDR_GREEN), &m_pPalette[3] );
m_PalNo[4] = LoadPalette( hInstance,
       MAKEINTRESOURCE(IDR_BLUE), &m_pPalette[4] );
```

いちばん上のコメント行が従来のロード方法である。それに対し，LoadDIBSection()という関数を使って，パレット部分とDIB部分を分離してロードしている。ここで分離されたパレットはシルバーのパレットである。残りのパレットは，リソースからLoadPalette()という関数で読み込んでいる(先ほど作ったパレットは，リソースタイプ名"PALETTE"とでもしてリソースにインポートしておく)。

ここのLoadDIBSection()とLoadPalette()は私が作ったユーザー関数であり，Palette.cppとPalette.hで宣言してあるので，それをプロジェクトに加え，Palette.hをインクルードしておくこと。こうしておいて，CMascotのメンバm_pPaletteInfoにパレットのインフォメーションを代入する。これはPALETTEINFOという構造体である。

```
typedef struct PALETTEINFO {
  int m_ColorMin, m_ColorMax;
  RGBQUAD * m_pColorPalette;
  int m_MaskMin, m_MaskMax;
  RGBQUAD * m_pMaskPalette;
} PALETTEINFO;
```

m_ColorMinとm_ColorMaxには，パレットを割り当てる最初のインデックスと最後のインデックスを指定する。この場合はm_ColorMinに0を，m_ColorMaxにLoadPalette()の戻り値であるm_PalNo[]を指定すればよい。m_pColorPaletteにはパレットの実体であるm_pPalette[]を指定，その下はマスクのパレットも同様に指定できるようにしてある。

さて，ドラゴンは最大10(MAXCHARNUM)出現し，1匹のドラゴンは最大20のパーツで構成されているとしよう。すると，キャラクター数はMAXCHARNUM×20必要である。それと，俯瞰図なので，画面の下にいるキャラクターほど手前に重ね合わされて表示してもらいたい。

あと，ドラゴンをマウスでクリックしたときになんらかのアクションもつけてみよう。こういったこともシェルがサポートしている。先ほどのCMascotDll::LoadChar()メソッドで次のようにリターンする。

```
return (MAXCHARNUM * 20)|BOTTOMSORT|HOOKCLICK;
```

本来この戻り値はキャラクターの数を返すものなのだが，オプションとしてソートとマウスクリックのフックを指定できる。BOTTOMSORTというのが，画面下のほうにいるキャラクターほど手前に描画するフラグである。もし画面上のほうを手前にしたければTOPSORT，同様にLEFTSORT，RIGHTSORTも用意されている。HOOKCLICKがクリックを取得するフラグであり，このフラグを指定しておいて，キャラクター上でマウスの左ボタンがクリックされると，CMascotDll::SetPattern()メソッドでCMascotのインスタンスm_NewShowのLBCLICKビットが立つ。変数の使い回しであまり美しくないが，互換性を持たせたまま拡張を行っていった代償である。やむなし。

さて，ドラゴンの動きであるが，こーゆーものは，頭だけが進路を決めて，胴体パーツは直前のフレームの，自分の前のパーツの動きをトレースするだけでよい。直前の座標などは，CMascotDll::SetPattern()が呼ばれた時点で，CMascotのインスタンスに残っていることが保証されているので，それを順にあと送りすればよい。したがって，座標などを保持する変数は頭パーツのMAXCHARNUM個だけということになる。

頭の制御は，先ほど述べた極座標で動作を決定する。ここでちょっと注意が必要なのは，CMascotDll::SetPattern()メソッドの戻り値だ。シーラカンスでは特になにも説明せずにm_numを返した。これは表示されている(可能性のある)キャラクター数であるが，実はそちらでは値を返す必要はない。この戻り値は，ソートされるキャラクターの数を示すものだからだ。もう少し正確にいうと，先頭から何個のキャラクターをソートするかを示している。したがって，CMascotの配列が10個あったとして，途中が非表示になっていたとしても，10個目が表示されていれば10を返さなければならない。そのため，通常はm_numを返しておけばよいのだが，このドラゴンの場合はm_numはドラゴンの(頭の)数を示しており，胴体を含めると1匹当たり最大20個のキャラクターからなっているので，m_num×20を返さなければならない。私も最初は気づかずに(というより，ソート数を返さねばならないことなどすっかり忘れていた)，正常にソートされずにしばらく悩んでしまった。気をつけるように。

最後はクリックされた場合のアクションである。なんとなくソー○リアンを思い出してしまったので，クリックされた部分がどかーんと爆発して，胴体が1段短くなるようにしようと思ったのだが，どんふり氏の要望により，頭をクリックした場合は1段長くなるというアクションも加えることにした。爆発に使ったアニメーションパターン画像は図15，16である。

まあ点数が現れるのは冗談としてだ，マスク画像を不審に思うだろう。これは決して誤りではない。真っ白な画像でいいのだ。これが半透明のテクニックその2である。実際にシェル内部で行われている描画の方法を示してみると，図17のようになっている。まず背景画像とマスク画像のANDを取り，キャラクターのある部分を抜く。そうしておいて，その上にカラー画像をORで重ねているのだ。もしここでANDを取らなければ(1とANDを取れば)，背景とキャラクターがそのままORで重ねられることになる。つまり「なんとなく合成っぽい」画像が得られるわけだ。正確な合成ではないが，アニメーションしていればそれなりにそれっぽく見える。

もう少しいえば，マスク画像をグレースケールで作れば(決してモノクロ2値である必要はない)部分的に合成っぽいアニメーションを作ったり，黄色いマスクを用意して背景のB値だけをマスクするといったテクニックも使える。思ったように制御できるかどうかは責任持てないが。

さっきはさらっと流したが，マスクに対してもパレットアニメーションできるのは，そういうわけだ。また，図16のマスクは，大きさもキャラクター1個分しかない。CMascotクラス側でカラー画像とマスク画像の座標を別々に設定できるため，複数のアニメーションで同じマスクを使う場合は，マスクをひとつにまとめることができる。

さて，クリックされたときのm_NewShowは先ほど述べたとおりだが，クリックしたときにLBCLICKフラグが乗るのはいちばん上にあるキャラクターだけではなく，複数のキャラクターがその場所に重なっていたら，すべ

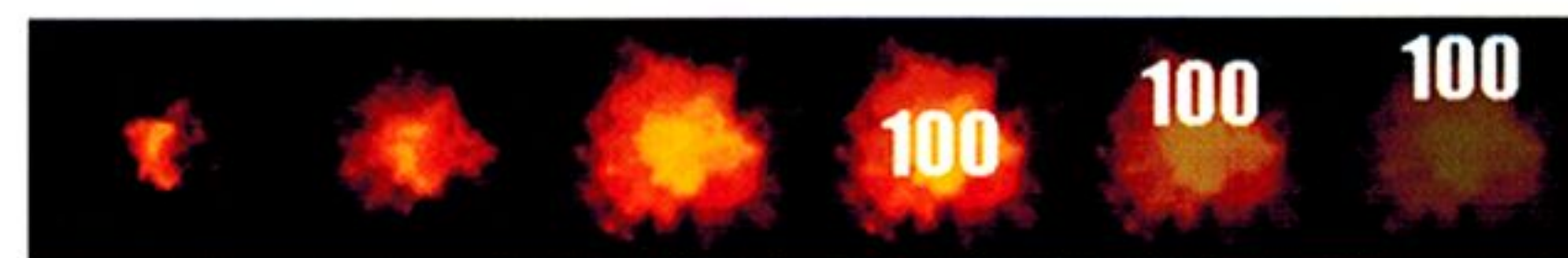



図15，16
爆発のアニメーションパターンとマスク画像

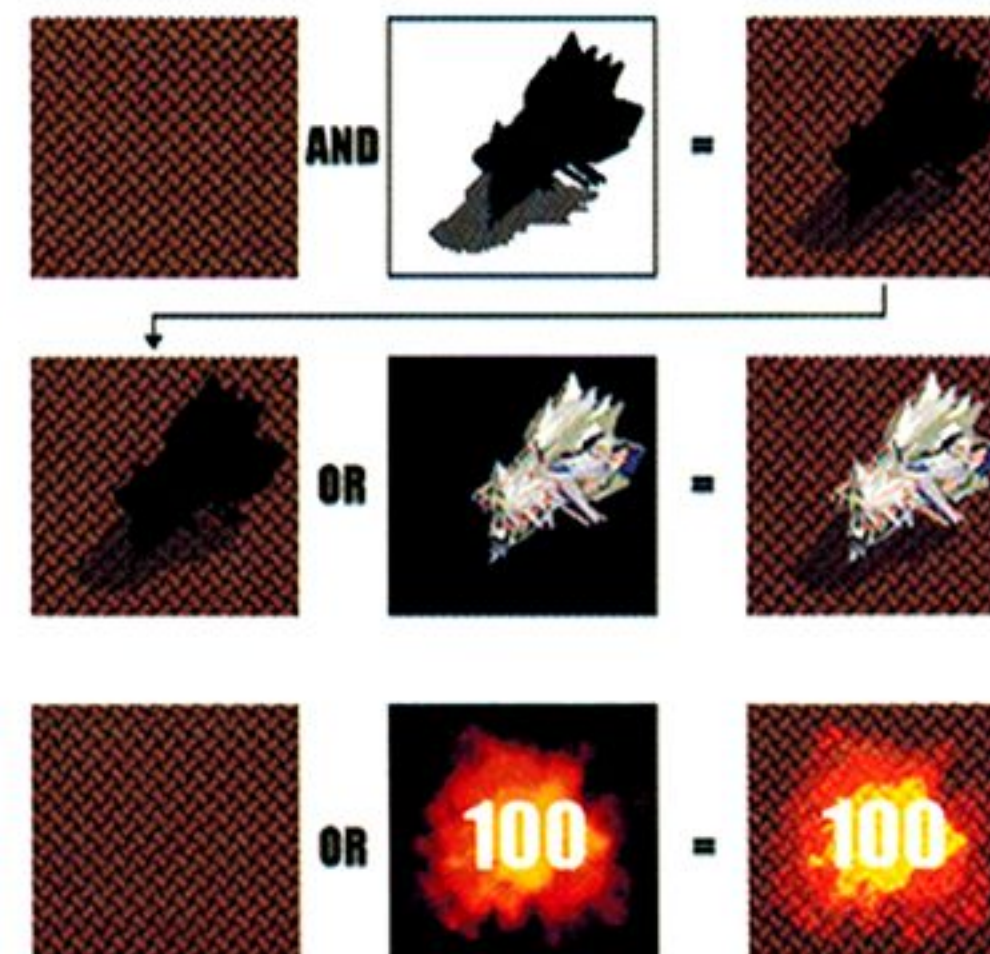



図17　背景とキャラクターの合成

図18, 19　豪快に爆発するメタルドラゴン

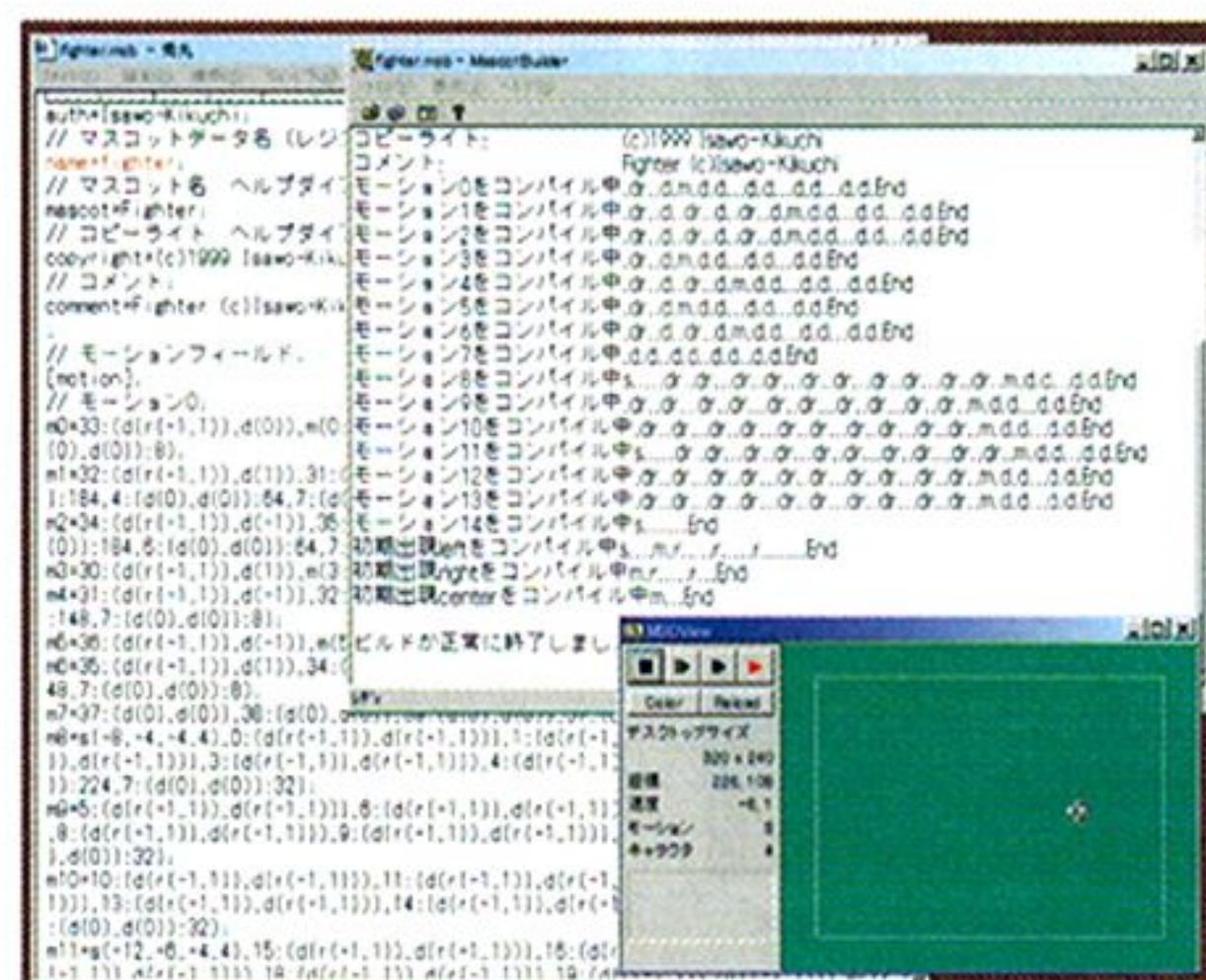



図20　現バージョンのMascotBuilderは純粋にソースをビルドするだけの機能しか持ち合わせていないが，シェルを起動させることなしにプレビューで動作確認はできる

てのキャラクターに対してLBCLICKフラグが乗る。特に今回のようにキャラクターが連なっている場合は，その可能性が大きい。一度のクリックで何段階も短くなるのは理不尽なので，先頭から検索し，クリックされていたら処理して検索を終了するようにしたほうがいいだろう。

1匹のドラゴンの長さはm_length[]で保持してあり，爆発のアニメーションはそれ以降のキャラクターを使用する。つまり，長さ5のドラゴンの先頭から3番目のパーツが爆発しても，爆発アニメーションは5番目のキャラクターが担当し，長さを1減らす。このときに効果音も鳴らすことにしよう。WAVファイルをリソースに加えたら,

```
PlaySound( MAKEINTRESOURCE(IDR_BOMB), hInstance,
                    SND_RESOURCE|SND_ASYNC );
```

のようにする。mmsystem.hをインクルードし，winmm.libをリンクに加えるのを忘れないように。

リストは主要部分だけでもずいぶん長くなってしまっているので，誌面では紹介できない。付録ディスクに収録されているサンプルDragonを参照してもらいたい。にょろにょろという動きもいいが，やはり爆発が熱い（図18, 19）。思わずクリックする手に力が入り，「うお～！首だけにしてやる～！」とムキになってしまいそうだ。ただ，あんまり熱中しすぎると，デスクトップアイコンがぐしゃぐしゃになっていたりするので気をつけよう。

Mascot Development Kit(MDK)編

「言語なんか知るか！」「Visual C++なんて高くて買えないよ」という人には，言語の知識を必要としない開発キットを用意した。このMDKは，4つのツールから構成されている。

・MascotBuilder
　マスコットデータをビルドするためのツール。MDKの中核をなすものである
・BindIcon
　BMPからアイコンファイルを生成するツール
・MSDView
　MascotBuilderで作成したデータをプレビューするツール
・TileBitmap
　BMPファイルをタイル状に連結するツール

このMascotBuilderによって作成されたマスコットデータは，拡張子が従来の"MSC"とは異なり"MSD"となる。このデータ形式は，ネイティブ(DLL)ではなく，中間コード形式となっている。よって，"MSC"と比べて，可能なことは制限されるが，ある程度の動きまでならば，十分実用になるはずだ。なお，実行スピードに関しては，理論上はネイティブよりも遅くなるはずだが，いまどきのCPUではその違いを体感できるほどではないだろう。また，MascotShellからは"MSC"も"MSD"も同じように扱えるため，ユーザーはそれらの違いを意識する必要はない。

"MSD"形式が"MSC"形式と比べて優れている点として，サイズが小さいということが挙げられる。これは，プログラム部分をシェル付属のMSDLoad.dllに任せているためと，独自形式でパックしているためにデータに隙間がないことによる。また，"MSC"の中身はDLL，つまりプログラムであるために，ウイルスに冒される危険性があるが，"MSD"は純粋な「データ」なので，その心配がない。

だが，残念ながら現バージョンのMascotBuilderは，マウスひとつでラクラク操作，というわけにはいかない。テキストエディタでソース（拡張子MSB）をちくちく書いて，それをビルドするという形になっている（図20）。将来的にはGUIを搭載したいと思っているが，それも現バージョンの反響次第だろう。とはいえ，モーションを記述するコマンドは数えるほどしかなく，慣れてしまえば，するすると思いどおりの制御コードを書けるようになるはずだ（カッコの数にうんざりしそうではあるが）。ビルドにはアイコンファイルが必要になるが，BindIconを使えばBMPからアイコンを作成することができるので，ほかにはグラフィックツールをひとつ用意してもらいたい。まあ，メモ帳とペイントだけでもなんとかならないでもないが。

とりあえず，MascotBuilder(MSD)の制限事項を挙げておく。

1.キャラクターは1枚の画像に収まっていなければならない
2.キャラクターのサイズは固定
3.キャラクターの拡大縮小機能は利用できない
4.後述の「モーションフィールド」で記述できない動きはつけられない
5.プロパティダイアログ，ヘルプボタンはない
6.キャラクターの数以外の設定項目はつけられない
7.データ間で同期を取ることはできない
8.パレットアニメーションはできない
9.マウスクリックに反応する機能は利用できない
10.その他，音を鳴らすなど独自の機能は一切つけられない

今回紹介したネイティブのサンプルでは，キャラクターサイズはすべて同じだったが，1コマずつ大きさを変えることができるのは説明せずともわかるだろう。MSDではそれが固定となる。また，データ間で同期を取れないというのは，ドラゴンのように複数のキャラクターに相関関係を持たせることができないということだと思ってもらいたい。ただし，このうちパレットアニメーションやサウンドは将来的には対応する可能性もある（予定はないが，実装は比較的簡単，という意味。要望が多ければ考えよう）。

MascotBuilderには，サンプルが2点同梱されている。まずはそちらをさっと眺めてみよう。

●スケルトン

例によってガイコツクラゲモドキである。リスト3はそのソースファイル（MSBファイル）だ。これを見てピンときた人もいるかもしれない。そう，これはイニシャルファイル(INI)形式なんである。'['，']'でくくられているのがセクション名，"エントリー名"="データ"という形で記述されている。ちなみにコメントは行の先頭に"//"をつけているが，実はエントリー名とぶつからなければこれは必要ない。ま，これは人間が見たときに，ここはコメントだよ，ということをわかりやすくしているにすぎない。逆にいえば，行の途中からコメントを書くことや，データの中に改行（スペースやタブも）を入れることは許されていない。また，ひとつのデータは最大1023文字までとしている。

識別子は触ってはならない。ここでbuilderversionが0.10となっているのは，このサンプルが旧版で作成されたからであり，現在は0.20となる。リソースフィールドはアイコンファイル，キャラクター画像ファイル，マスク画像ファイルの指定である。アイコンファイルは先ほど述べたように，

図21　ソースファイルのエントリと表示部分の対応

図22　使用されるレジストリ

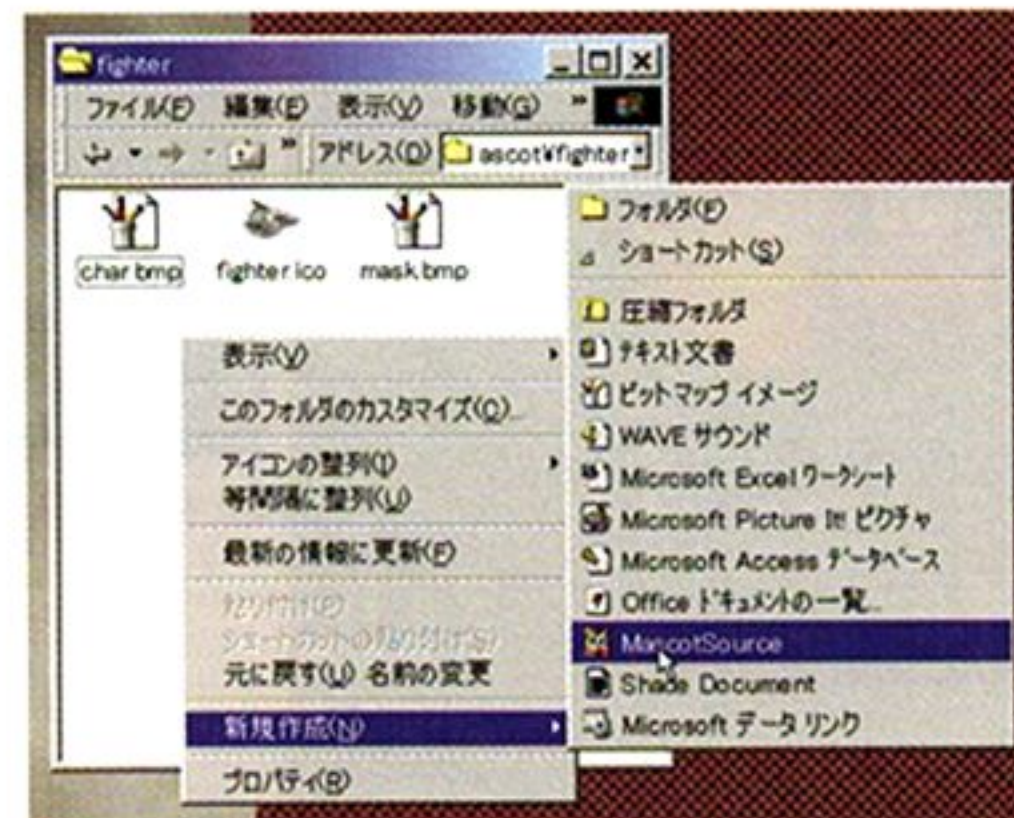



図23　キャラクターナンバー。画面右端にサイズに満たない端数があっても切り捨てられる

BindIconで作成したファイルを指定する（BindIconの使用法については，そちらのヘルプを参照のこと）。もちろん別途アイコンを作るツールを持っているのならば，そちらで作成したものでも構わない。キャラクター画像とマスク画像については，基本的にネイティブのほうで使ったものと同じだ。ただし，必ず同じサイズにしなければならない（ドラゴンの爆発でやったように，キャラクターとマスクで別々の座標を指定できない）。また，これらの画像は必ずMSBファイルと同じフォルダに入れておくこと。

マスコットデータフィールドは，キャラクターのプロパティや著作権情報を格納する。キャラクターソートはこのソースではONが1，OFFが0となっているが，version 0.20からは，

OFF:0 BOTTOM:1 TOP:2 RIGHT:3 LEFT:4

となった。それぞれネイティブのソート方式で説明した意味である。バージョンは，MSDの場合はそのときのシェルのバージョンにこだわらず，好きにつけていいことにした。これ以降の文字列は，それぞれ図21の部分で表示される部分と，図22のレジストリ部分で使用される。

いよいよモーションフィールドである。MascotBuilderのモーションは，細かいモーションデータを設定し，それを繋いで再生するという形をとっている。各モーションはモーションフィールドにm0，m1，m2，……といったように通し番号をつけたエントリに記述する。必ず0からの連番で指定すること。桁が増えた場合も"m10"といった表記で構わないが，桁数を揃えようとして"m00"などとしても通らない。

初期出現のtop，bottom，left，right，centerエントリも（このサンプルではbottomしか使っていないが），特殊なモーションエントリと考えることができる。つまり，マスコットが最初に出現するポイントを指定している。topは画面上端，bottomは画面下端，rightは右端，leftは左端からの出現になり，centerはそれ以外，つまり画面内の任意の位置から出現する場合となる。この5つのエントリのうち，少なくともひとつは設定しないことにはマスコットは出現できず，ビルド時にエラーとなる。

コラムで解説されているコマンドを踏まえて，先ほどのソースをざっと読み下してみよう。まず，初期出現は画面の下端からだ。臨界速度はX方向に－2から2，Y方向に－10から6となっている。つまり，上向きに進むことが予想できる（画面下端から出現するのだから当たり前だが）。まずはモーション0に接続する。このとき，速度はX方向に－1から1，Y方向に－2から－6の範囲でランダムに与えている。

モーション0ではキャラクターナンバー0，1，2をアニメーションしているが，1と2の表示時にY方向の速度として－1から0の範囲で加速度を与えている。つまり，なんとなく上向きに加速しているということだ。そのあと，半分の確率で同じモーション0をループするか，モーション1に分岐している。モーション1では，3フレーム間キャラクターナンバー2を表示させており，その間にYの速度を下向きに1だけ加算している。分岐は8/256の確率で初速を－2から－6の範囲で与えてモーション0に，それ以外であればモーション1をループしている。だいたいどんな動きをするかイメージできただろうか。画面下から現れたキャラクターが，必死に上に向かって泳いでいくが，ちょっと泳いでは疲れて休んでしまうため，重力によって落ちていきそうになる，というのの繰り返しとなる。

●飛行機

上のスケルトンのようなものならば問題ないのだが，XとYを独立して指定するとなると，方向というものを制御することができない。たとえば，虫が8方向に歩き回るといった場合だ。そのようなもののために，version 0.20からは極座標モードを搭載した。ネイティブでやったドラゴンの座標系と同じだ。その極座標モードのサンプルとして，MascotBuilderにはPlaneも同梱してある（リスト4）。ソースを見ると，座標モードというエントリが増えていることがわかるだろう。ここで1を指定すると，極座標モードでモーションを記述できる。

コマンド自体は先ほど説明したものと同じものが使えるが，Xの代わりに角度θ，Yの代わりに距離（速度）Lを指定する。θは上向きが0時計回りが正の方向で，$2\pi/64$単位である。つまり，キャラクターの動きは64方向まで指定できるのだが，キャラクター自体は64方向のパターンを作成しなく

リスト3　スケルトンの例

```
// 識別子
[generic]
datatype=MascotBuilder
buildversion=0.10

// リソースフィールド
[res]
// アイコンファイルの指定
icon=mascot.ico
// キャラクタ画像の指定
char=char.bmp
// マスク画像の指定
mask=mask.bmp

// マスコットデータフィールド
[mascot]
// キャラクタの大きさ
size=32,32
// キャラクタの最大出現数
maxnum=20
// デフォルトのキャラクタ数
defnum=4
// キャラクタソート (ON:1 OFF:0)
sort=0
// バージョン
version=1.00
// 著作権者 (レジストリキーで使用)
auth=Isawo-Kikuchi
// マスコットデータ名 (レジストリキーで使用)
name=Skeleton
// マスコット名　ヘルプダイアログで表示
mascot=スケルトン
// コピーライト　ヘルプダイアログで表示
copyright=(c)1999 Isawo-Kikuchi
// コメント
comment=ﾏｽｺｯﾄﾃﾞｰﾀｻﾝﾌﾟﾙ ｽｹﾙﾄﾝ

// モーションフィールド
[motion]
// モーション0
m0=0:(d(0),d(0)),1:(d(0),d(r(-1,0))),2:(d(0),d(r(-
1,0))),m(0:(d(0),d(0)):128,1:(d(0),d(1)):128)
// モーション1
m1=2:(d(0),d(0)),2:(d(0),d(1)),2:(d(0),d(0)),m(0:(d(0),r(-2,-
6)):8,1:(d(0),d(1)):248)
// 初期出現
bottom=s(-2,2,-10,6),m(0:(r(-1,1),r(-2,-6)))
```

てもよい。

ここで重要なのが，最後のエントリのキャラクターナンバーオフセットである。このエントリでは，64方向に対するキャラクターナンバーのオフセット値を指定する。このサンプルでは，飛行機のパターン自体は16方向であるので，それぞれのパターンに対して方向を64/16＝4個ずつ割り振っている。また，この値はあくまでもオフセットであるので，モーションデータで指定されたキャラクターナンバーに，そのときの方向に対するこの値が加算されたものが，実際に表示されるキャラクターとなる。したがって，アニメーションパターンの記述も可能だ。

モーションデータ自体はシンプルにしてある。角速度は-2から2，速度は4から16の範囲で制限しており，それぞれランダムに-1から1の範囲で値を加算しているだけだ。だいたいどんな動きをするかも容易に想像できるだろう。なお，初期出現時の向きは，出現位置で決まっている。画面上端からの出現ならば，下方向，つまり角度でいうところの17から47までの範囲のランダム値，左端ならば1から31までといった具合だ。画面内の任意の位置(center)の場合は，全方向が対象となる。

●Fighter

だいたいのことはわかっていただけたと思うので，実際の作成法に沿ってひとつ作ってみよう。まずはなにはともあれアイコンと，キャラクターとマスク画像が必要だ。これを見ると，戦闘機3種＋爆発の4種類のキャラクターが混在している。MSDの制限で「画像はひとつだけ」といったが，反対にいえば1枚の画像に収めてしまえば複数のキャラクターを収録することも可能ということだ。また，爆発の部分は，ドラゴンでもやったエセ半透明テクニックを使っている。ではこのキャラクターを動かすソースを書いてみよう。

リスト4　飛行機の指定例

```
// 識別子
[generic]
datatype=MascotBuilder
buildversion=0.20
// リソースフィールド
[res]
// アイコンファイルの指定
icon=mascot.ico
// キャラクタ画像の指定
char=char.bmp
// マスク画像の指定
mask=mask.bmp

// マスコットデータフィールド
[mascot]
// キャラクタの大きさ
size=32,32
// キャラクタの最大出現数
maxnum=20
// デフォルトのキャラクタ数
defnum=4
// キャラクタソート (OFF:0 BOTTOM:1 TOP:2 RIGHT:3 LEFT:4)
sort=0
// 座標モード (直行座標:0 極座標:1)
mode=1
// バージョン
version=1.00
// 著作権者 (レジストリキーで使用)
auth=Isawo-Kikuchi
// マスコットデータ名 (レジストリキーで使用)
name=Plane
// マスコット名　ヘルプダイアログで表示
mascot=飛行機
// コピーライト　ヘルプダイアログで表示
copyright=(c)1999 Isawo-Kikuchi
// コメント
comment=ﾏｽｺｯﾄﾃﾞｰﾀｻﾝﾌﾟﾙ ﾌﾟﾚｰﾝ

// モーションフィールド
[motion]
// モーション0
m0=0:(d(0),d(0)),m(0:(d(r(-1,1)),d(r(-1,1))))
// 初期出現
top=s(-2,2,4,16),m(0:(r(-2,2),r(4,16)))
bottom=s(-2,2,4,16),m(0:(r(-2,2),r(4,16)))
left=s(-2,2,4,16),m(0:(r(-2,2),r(4,16)))
right=s(-2,2,4,16),m(0:(r(-2,2),r(4,16)))
// キャラクタナンバーオフセット
offset=0,0,1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6,7,7,7,7,8,8,8,8,9,9,9
,9,10,10,10,10,11,11,11,11,12,12,12,12,13,13,13,13,14,14,14,14,15,15,15,15,0,0
```

まだ一度もMascotBuilderを起動していないのなら，一度立ち上げてもらいたい。過去に一度でも起動したことがあるのなら，そのままでよい。適当なフォルダの中でマウスを右クリックして，コンテキストメニューを表示しよう。その[新規作成]の中に，"MascotSource"というのが増えているはずだ。それを選択すると，リスト5のようなスケルトンが生成されるので，それに書き込んでいけばよい。

マスコットデータフィールドまではいいだろうから，Fighterのモーションフィールドだけを抜粋しよう(リスト6)。はっきりいって説明するのもうんざりしてしまうし，こまごまと解説しても読む気にならないだろうから，だいたいの構成だけを説明する。

初期出現で左から出現するのは，画像内で下から2行の中にある右向きの戦闘機である。モーション0，1，2，14に分岐しているが，0は水平飛行，1は機体を右向きに，2は左向きに傾けるアニメーションである。14のほうはというと，単に出現数を減らすだけのダミーであり，画面に表示されることなしに消えてしまう。

さらに1は3と4に，2は5と6に分岐しているが，これは傾いたまま飛行するパターンと，傾いた状態から水平状態に戻るパターンである。ここまでの7つのモーションはその中でだいたい自己完結しているが，ときおり分岐している。モーション7は爆発のパターンである。モーション7は最後にモーション分岐コマンドがないので，爆発したあとは消滅する。それに対し，画面右から出現するのは，左向きの戦闘機2種であり，モーション8への分岐が丸っこいほう，11への分岐はとがっているほうだ。丸っこいほうのモーションは8から10まで，とがったほうは11から13までであり，アニメーションとしては回転するだけなのだが，すべてを同じモーションに記述するとエントリが長くなってしまうので，3つに分けているにすぎない。

ときどきモーション7に分岐しているのは，さっきと同様爆発パターンである。最後に，画面の任意の領域での出現は，モーション7，すなわち爆発に分岐している。唐突になにもなかったところで爆発を起こしてみた。

ソースが書けたら，ビルドしてみよう。もしMascotBuilderが立ち上がっていたら，そこにソースをドロップするか，[ファイル][開く]でソースを選択する。それだけでビルドが始まり，メッセージがだらだらと流れる。ソースが適切であったなら，最後に「ビルドが正常に終了しました」と表示されるはずだ。もしなにかしらのエラーがあったら，その旨メッセージが出る。モーションデータでエラーが発生した場合には，図24のような感じになる。モーションデータのコンパイル中は，数値が'.'，d，r，m，sコマンドがそれぞれ'd'，'r'，'m'，'s'で表示されるので，エラーの出た個所をだいたい特定できるはずだ。修正したら，今度は[ファイル][ビルド]を選択するか，ツールバーの左から2番目のビルドボタンを押すだけでよい。エラーが消えるまでこれを繰り返すことになる。

さて，無事にビルドができても，意図した動作になっているかは動かしてみないとわからない。シェルを立ち上げてロードしてみてもいいのだが，MSDViewを使えばもっと手軽に動作を確認できる。もしMascotBuilderとMSDViewを同じフォルダに入れているのなら，[ファイル][プレビュー]を選択，あるいはツールバーの3番目のプレビューボタンを押すだけで，MSDViewが立ち上がり，いまビルドしたデータをプレビューしてくれる

MSDView
Color　Reload
デスクトップサイズ　320 x 240
座標　62, 85
速度　8, 4
モーション　3
キャラクタ　30

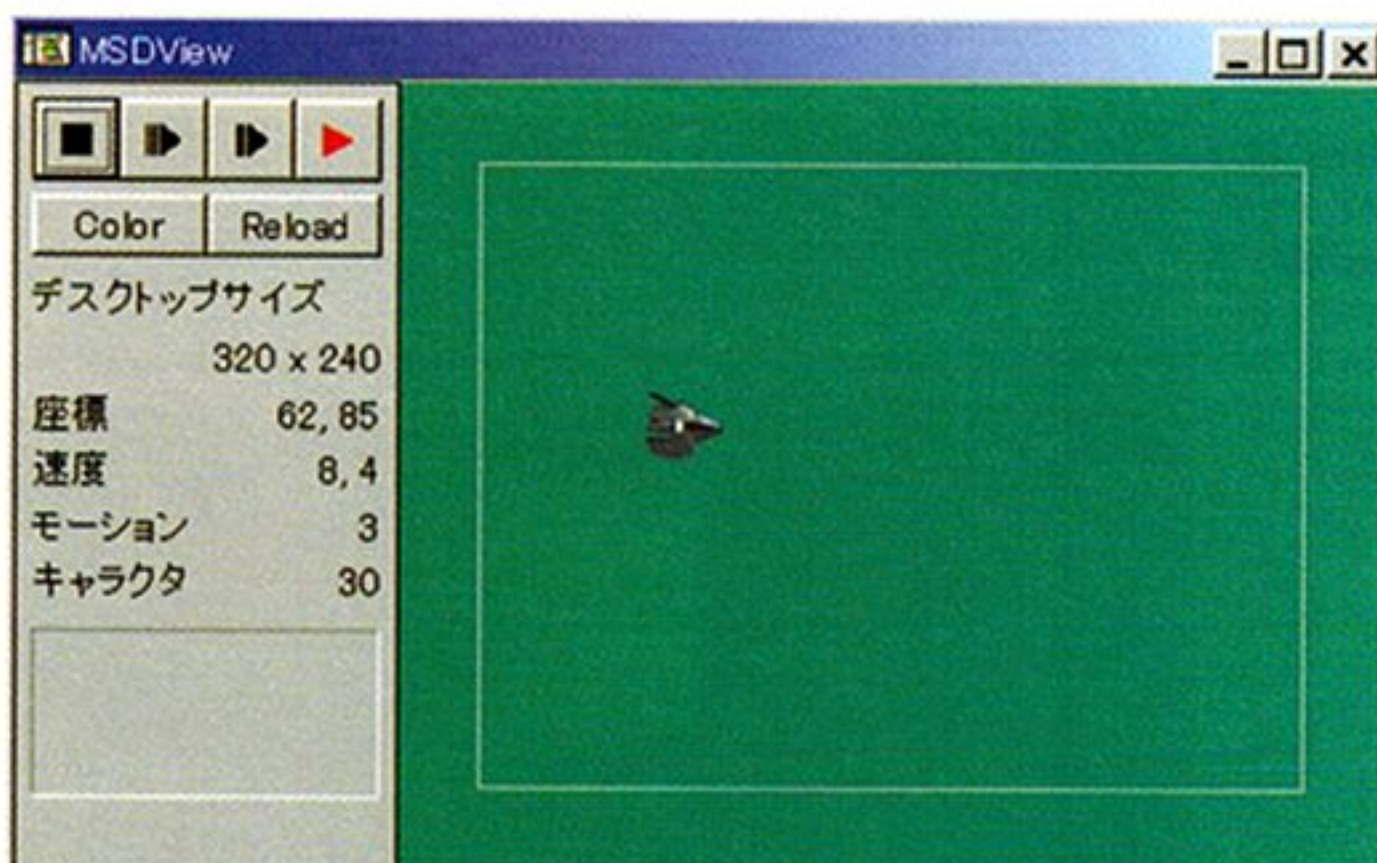



図24　情報をリアルタイムで表示するので，モーションデータの確認ができる。枠のマージン部分で出現ポイントを確認できるのもミソ

リスト5 Fighter用スケルトン

```
// 識別子（ここは編集しないでください）
[generic]
datatype=MascotBuilder
buildversion=0.20

// リソースフィールド
[res]
// アイコンファイルの指定
icon=
// キャラクタ画像の指定
char=
// マスク画像の指定
mask=

// マスコットデータフィールド
[mascot]
// キャラクタの大きさ
size= ,
// キャラクタの最大出現数
maxnum=
// デフォルトのキャラクタ数
defnum=
// キャラクタソート（OFF:0 BOTTOM:1 TOP:2 RIGHT:3 LEFT:4）
sort=
// 座標モード（直行座標:0 極座標:1）
mode=
// バージョン
version=
// 著作権者（レジストリキーで使用）
auth=
// マスコットデータ名（レジストリキーで使用）
name=
// マスコット名 ヘルプダイアログで表示
mascot=
// コピーライト ヘルプダイアログで表示
copyright=(c)1999
// コメント
comment=

// モーションフィールド
[motion]
// モーション0
m0=

// 初期出現（不要な箇所は削除してください）
top=
bottom=
left=
right=
center=
// キャラクタナンバーオフセット（極座標モードのみ）
offset=
```

リスト6 モーションデータ

```
// モーションフィールド
[motion]
// モーション0
m0=33:(d(r(-1,1)),d(0)),m(0:(d(0),d(0)):232,1:(d(0),d(0)):8,2:(d(0),d(0)):8,
   7:(d(0),d(0)):8)
m1=32:(d(r(-1,1)),d(1)),31:(d(r(-1,1)),d(1)),30:(d(r(-1,1)),d(1)),
   m(3:(d(0),d(0)):184,4:(d(0),d(0)):64,7:(d(0),d(0)):8)
m2=34:(d(r(-1,1)),d(-1)),35:(d(r(-1,1)),d(-1)),36:(d(r(-1,1)),d(-1)),
   m(5:(d(0),d(0)):184,6:(d(0),d(0)):64,7:(d(0),d(0)):8)
m3=30:(d(r(-1,1)),d(1)),m(3:(d(0),d(0)):232,4:(d(0),d(0)):16,7:(d(0),d(0)):8)
m4=31:(d(r(-1,1)),d(-1)),32:(d(r(-1,1)),d(-1)),m(0:(d(0),d(0)):100,
   2:(d(0),d(0)):148,7:(d(0),d(0)):8)
m5=36:(d(r(-1,1)),d(-1)),m(5:(d(0),d(0)):232,6:(d(0),d(0)):16,7:(d(0),d(0)):8)
m6=35:(d(r(-1,1)),d(1)),34:(d(r(-1,1)),d(1)),m(0:(d(0),d(0)):100,
   1:(d(0),d(0)):148,7:(d(0),d(0)):8)
m7=37:(d(0),d(0)),38:(d(0),d(0)),39:(d(0),d(0)),37:(d(0),d(0))
m8=s(-8,-4,-4,4),0:(d(r(-1,1)),d(r(-1,1))),1:(d(r(-1,1)),d(r(-1,1))),
   2:(d(r(-1,1)),d(r(-1,1))),3:(d(r(-1,1)),d(r(-1,1))),4:(d(r(-1,1)),
   d(r(-1,1))),m(9:(d(0),d(0)):224,7:(d(0),d(0)):32)
m9=5:(d(r(-1,1)),d(r(-1,1))),6:(d(r(-1,1)),d(r(-1,1))),7:(d(r(-1,1)),
   d(r(-1,1))),8:(d(r(-1,1)),d(r(-1,1))),9:(d(r(-1,1)),d(r(-1,1))),
   m(10:(d(0),d(0)):224,7:(d(0),d(0)):32)
m10=10:(d(r(-1,1)),d(r(-1,1))),11:(d(r(-1,1)),d(r(-1,1))),12:(d(r(-1,1)),
    d(r(-1,1))),13:(d(r(-1,1)),d(r(-1,1))),14:(d(r(-1,1)),d(r(-1,1))),
    m(8:(d(0),d(0)):224,7:(d(0),d(0)):32)
m11=s(-12,-6,-4,4),15:(d(r(-1,1)),d(r(-1,1))),16:(d(r(-1,1)),
    d(r(-1,1))),17:(d(r(-1,1)),d(r(-1,1))),18:(d(r(-1,1)),d(r(-1,1))),
    19:(d(r(-1,1)),d(r(-1,1))),m(12:(d(0),d(0)):232,7:(d(0),d(0)):24)
m12=20:(d(r(-1,1)),d(r(-1,1))),21:(d(r(-1,1)),d(r(-1,1))),22:(d(r(-1,1)),
    d(r(-1,1))),23:(d(r(-1,1)),d(r(-1,1))),24:(d(r(-1,1)),d(r(-1,1))),
    m(13:(d(0),d(0)):232,7:(d(0),d(0)):24)
m13=25:(d(r(-1,1)),d(r(-1,1))),26:(d(r(-1,1)),d(r(-1,1))),27:(d(r(-1,1)),
    d(r(-1,1))),28:(d(r(-1,1)),d(r(-1,1))),29:(d(r(-1,1)),d(r(-1,1))),
    m(11:(d(0),d(0)):232,7:(d(0),d(0)):24)
m14=s(0,0,0,0),33:(0,0)
// 初期出現（不要な箇所は削除してください）
left=s(4,8,-4,4),m(0:(r(4,8),0):64,1:(r(1,8),0):32,2:(r(1,8),0):32,14:(0,0):128)
right=m(8:(r(-8,-4),0):192,11:(r(-12,-6),0):64)
center=m(7:(0,0))
```

図25 エラーが発生した場合は，適切なメッセージが表示される，はずなのだが，こういうものは時として的外れなアドバイスとなることも多い

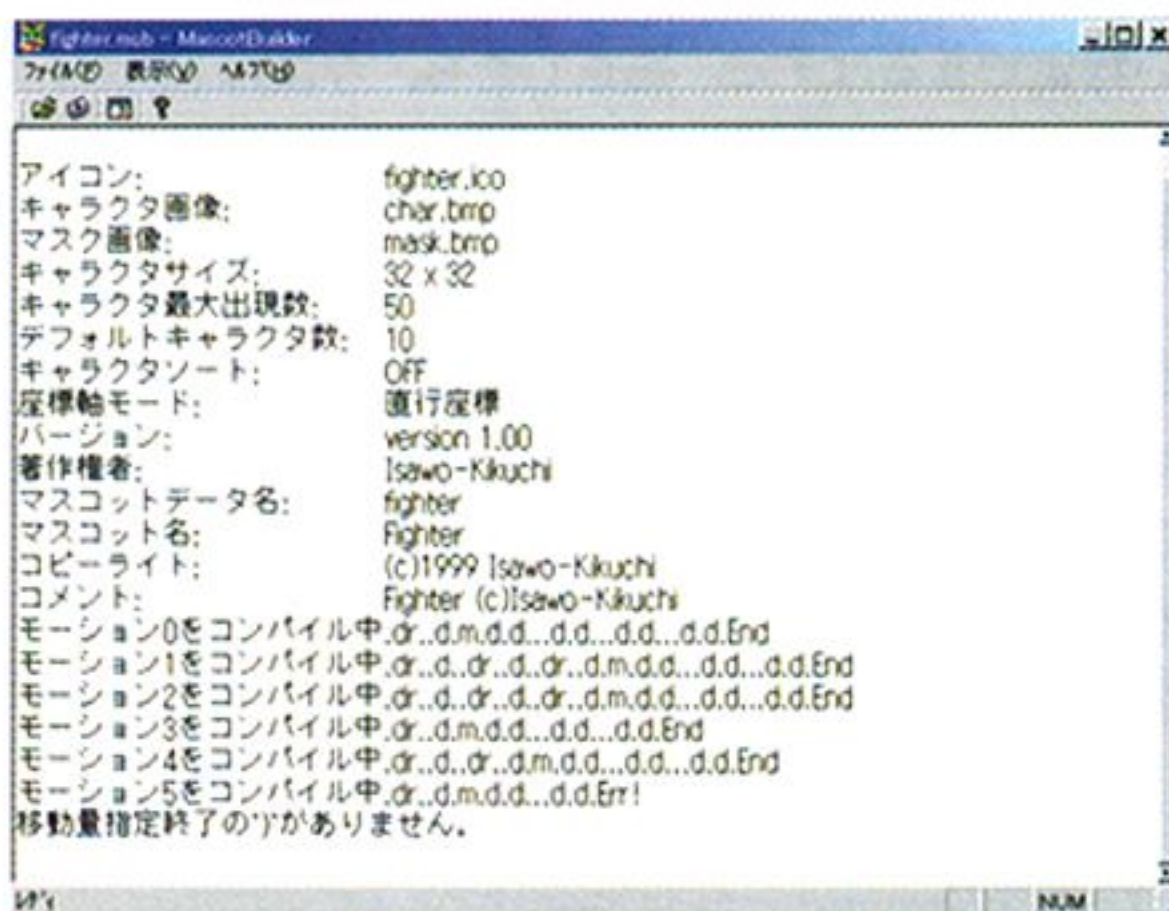



図26 戦闘の渦中に飛び込んでしまったデスクトップ。なにもしないのに戦闘機が爆発するのは，武器がレーザーだからららしい

(図25)。このMSDViewはキャラクターをひとつしか表示しないが，座標や速度，モーションナンバー，キャラクターナンバーをリアルタイムで表示し，スロー，コマ送りもできるようになっているので，デバッグには重宝するだろう。ぜひ利用してもらいたい。

モーションに納得いったら完成だ(図26)。木の葉のような戦闘機が，両軍入り乱れて戦闘を行うデータとなった。アイコンや画像は生成されたMSDファイル内にパックされているので，配布するのはMSDファイルと，あとドキュメントでもつければいいだろう。ドキュメントには，動作が確認できているシェルのバージョンと，シェルの所在地を忘れずに明記しておくこと。

データが完成したら

完成したデータは，MSC，MSDともに配布は自由である。ホームページに掲載するなりコミケで売るなり好きにして構わない。ただし，シェルも同時に配布する場合には，あらかじめ私のほうまで連絡し，確認を取ってほしい。ソースの配布も自由だが，最初にいったように，ライブラリやMSDの添付は認めない。ひとつ注意してもらいたいのは，レジストリで使用される著作権者名には，ありがちなハンドルなどは避けるべきである。BackgroundMascotShellが複数のデータをロードできるということは，異なる作者によるデータも同時に使用する可能性があるということだ。その場合，もしレジストリが干渉すれば，正常な動作の妨げとなる。

BackgroundMascotShellの最新版はhttp://www.hinix.com/kisawo/に掲載してある。また，マスコットデータは作者を問わずhttp://www.hinix.com/kisawo/mascot/にて展示されている。もし自分で作成したデータをここに掲載したい，あるいはリンクを張ってほしいというのであれば，私まで連絡いただきたい(モノによっては遠慮いただく場合もあるかもしれないが)。また，そうでなくてもデータを作成した旨を報告いただければ，不具合がないかなどのチェックを行うことは可能であるし，作者としてもどんなものが作成されているか把握しておきたいところでもある。同Webページに掲示板を設けてサポートを行う予定であるので，そちらも定期的にチェックしておくとよいだろう。

MascotBuilder version 0.20　簡易マニュアル

【インストール】
インストールは解凍のみで，他に必要な作業は特にありません。最初に Mascot-Builder を起動した時点で，各種設定をレジストリに書きこみますので，まずはなにはともあれMascotBuilder.exeを起動してください。

【アンインストール】
解凍して生成したファイルと，レジストリの以下の項目を削除してください。

```
HKEY_CLASSES_ROOT¥.msd
HKEY_CLASSES_ROOT¥BGMascot.msd
HKEY_CURRENT_USER¥Software¥X68k Users Group¥MascotBuilder
```

【使い方】
まずはソースを書いてください。すでに一度でもMascotBuilderを起動してレジストリの設定が行われていれば，右クリックして表示されるコンテキストメニューの[新規作成]-[MascotSource]で新規スケルトンソースを作成できます。あるいは，MascotBuilder.exeと同じフォルダにインストールされたnew.msbをコピーしてもかまいません。テキストエディタで開くと，次のように記述されています。

```
// 識別子(ここは編集しないでください)
[generic]
datatype=MascotBuilder
buildversion=0.20

// リソースフィールド
[res]
// アイコンファイルの指定
icon=
// キャラクター画像の指定
char=
// マスク画像の指定
mask=

// マスコットデータフィールド
[mascot]
// キャラクターの大きさ
size= ,
// キャラクターの最大出現数
maxnum=
// デフォルトのキャラクター数
defnum=
// キャラクターソート(OFF:0 BOTTOM:1 TOP:2
                    RIGHT:3 LEFT:4)
sort=
// 座標モード(直行座標:0 極座標:1)
mode=1
// バージョン
version=
// 著作権者(レジストリキーで使用)
auth=
// マスコットデータ名(レジストリキーで使用)
name=
// マスコット名　ヘルプダイアログで表示
mascot=
// コピーライト　ヘルプダイアログで表示
copyright=(c)1999
// コメント
comment=

// モーションフィールド
[motion]
// モーション0
m0=

// 初期出現(不要な箇所は削除してください)
top=
bottom=
left=
right=
center=
// キャラクターナンバーオフセット(極座標モードのみ)
offset=
```

コメントのとおりに各フィールドを埋めてください。MascotBuilderをインストールしたフォルダの中のSampleフォルダにあるサンプルデータを参考にしてもいいでしょう。モーションフィールドについてはかなり複雑ですので，後述します。

[キャラクター画像とマスク画像]
キャラクター画像とマスク画像は必ず同じドットサイズにしてください(キャラクター画像とマスク画像の内容については，Sampleフォルダ内のchar.bmpとmask.bmpを参考にしてください)。キャラクター画像の背景部分は，必ず黒にしてください。また，マスク画像はモノクロ2値，キャラクター画像はサイズのことも考慮して，4ビットか8ビットをRLE圧縮したものを推奨します。

[キャラクターソート]
同一データ内のキャラクター同士であれば，画面上で重なった場合にその位置関係によって，どちらを手前に表示するか指定することができます。sortに1(BOTTOM)を指定すると画面上で下のほうにいるキャラクターほど手前に表示されます。TO_P,RIGHT,LEFTも同様です。

[座標モード]
モーションフィールドのデータは，右向きをX，下向きをYとしたXY直行座標と，真上を0とした時計回りの極座標で記述できます。modeに0を指定すると直行座標，1を指定すると極座標となります。

ソースが書けたら，MascotBuilderでビルドします。レジストリに設定がされていればmsbファイルをダブルクリックすることで，自動的にMascotBuilderが立ち上がってビルドが行われます。すでにMascotBuilderが立ち上がっていれば，MascotBuilderからファイルを開くか，msbファイルをドロップすることでもビルドできます。もしエラーがなければ「ビルドが正常に終了しました。」と表示され，msdファイルが生成されているはずです。エラーが発生したら，エラー箇所をエディタで修正してください。このときにMascotBuilder側でドキュメントを閉じる必要はありません。修正ができたら再びビルドですが，今度はMascotBuilderのツールバーの，左から2番目の「ビルド」ボタンでビルドが行えます。ビルドが正常に終了するまでこれを繰り返します。
ビルドが終了したら，MascotShellに登録する前に，プレビューして動作を確認するといいでしょう。MSDViewがMascotBuilderと同じフォルダにインストールされていれば，[ファイル]メニューの[プレビュー]，あるいはツールバーの3番目の「プレビュー」ボタンで，いまビルドしたデータをプレビューして見ることができます。

【モーションフィールド】
MascotBuilderのモーションは細かいモーションデータを設定し，それを繋いで再生するという形をとっています。各モーションはモーションフィールドにm0,m1,m2,……といったように通し番号をつけたエントリに記述します。必ず0からの連番で指定してください。桁が増えた場合も"m10"といった表記でかまいませんが，桁数を揃えようとして"m00"などとしても通りません。初期出現のtop,bottom,left,right,centerエントリも，特殊なモーションエントリと考えることができます。つまり，マスコットが最初に出現するポイントを指定します。topは画面上端，bottomは画面下端，rightは右端，leftは左端からの出現になります。centerはそれ以外，つまり画面内の任意の位置から出現する場合です。この5つのエントリのうち，必ずひとつは設定しないことには，マスコットは出現できません。
モーションフィールドのフォーマットを語るうえで，まずは必要な用語を説明します。

・キャラクターナンバー
指定したキャラクター画像(マスク画像)内に，キャラクターサイズで指定された大きさの画像がマトリクス状に配置されているものと考えて，左上からキャラクター番号0，右向きに1番，2番と数えていき，いちばん右までいったらひとつ下のいちばん左からまた右へと数えていきます。
たとえば，画像が128×128，キャラクターサイズが32×32であれば，次のようになります。

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

このとき，画像の右端にサイズに満たない端数があったとしても切り捨てられます。

・モーションナンバー
モーションエントリーで付与された番号です。m0エントリーに記述されたモーションデータは，モーションナンバー0番ということになります。

・速度
前回(0.1秒前)からのキャラクターの移動ドット数を示します。画面右向きがXの正方向，画面下向きがYの正方向です。また，極座標モードでは，前回からの角度と移動量の変移を示します。

・加速度
前回(0.1秒前)からのキャラクター速度の変移を示します。極座標モードでは，角速度と移動速度の変移を示します。

・臨界速度
キャラクターが取り得る最大の速度を示します。

モーションデータの基本は，次の形です。

num:(x,y)
numはキャラクターナンバー，xはX方向の速度，yはY方向の速度を示します。このセットで0.1秒分を示し，これをカンマ','で繋げることで，連続したアニメーションデータを表現します。このとき，速度には次のコマンドも指定できます。

d(n)
これは速度を直接指定するのではなく，加速度nを与える表記です。つまり，直前の速度がdxだとすると，この時の速度はdx+nとなります。滑らかにアニメーションするキャラクターを作る場合，むしろ加速度表記のほうが重要でしょう。

また，速度および加速度には，ランダム性を持たせる次のコマンドを指定できます。

r(min,max)
minからmaxまでの値をランダムで返します。マイナス値はminやmax側で指定してください。つまり，r(-4,-1)は可ですが，-r(1,4)は不可です。

次のモーションへの連結は，モーションデータの最後にカンマで区切って次のように記述します。

m(n0:(mx0,my0):p0,n1:(mx1,my1):p1,…)
先頭のmはコマンドですので，このまま記述してください。n0,n1,……は，モーションナンバーを示し，(mx0,my0),(mx1,my1),……は次のモーションへ移行するときの速度を上と同様に指定します(dやrコマンドが使えます)。これは，いままで右を向いて右向きに歩いていたキャラクターが，反転して左を向いた場合などに利用できます。もし速度を維持したいのならばd(0)を指定してください。p0,p1,……は，そのモーションへの分岐確率を示します。それぞれ1/256単位ですので，すべての和が256を超えないようにしてください。反対に256に満たない場合，いちばん最後に記述されたモーションナンバーがデフォルトになります(つまり，いちばん後ろに記述される分岐確率はあまり意味がない)。もし分岐が必要ない(連結モーションナンバーはただひとつに固定されている)場合でも，mの後ろのカッコは省略できませんが，分岐確率(とその前のコロン)は省略できます。また，もしこのmコマンドがモーションデータの最後で指定されなかった場合，および存在しないモーションナンバーを指定した場合は，そこでキャラクターは消えます。mコマンド以降に記述されたデータは実行されません。
適当なところでカンマで区切り，以下のコマンドを挿入することで，臨界速度を指定できます。

s(xmin,xmax,ymin,ymax)
それぞれXの最小，最大，Yの最小，最大を示します。ここでいう最小・最大というのはあくまでも数値的な範囲の話で，たとえば左向きや上向きの最大速度を設定したければ，xminやyminにマイナス値を設定することになります。sコマンドで設定した臨界速度は，次にsコマンドが現れるまで有効です。なお，sコマンド内部でdやrコマンドは使えません。

極座標モードでは，上の説明中のXを角度θ，Yを距離Lと読み替えてください。θは真上が0，時計回りが正の方向で，2π/64単位です。また，極座標モードでは，ほかにキャラクターナンバーオフセットが必要です。これは，θの角度によってモーションフィールドで記述したキャラクターナンバーに加算される値で，offsetにカンマで区切って64個のオフセットを指定します。サンプルPlaneを参考にしてください。

コンパイル中は，数値が'.'，d,r,m,sコマンドがそれぞれ'd','r','m','s'で表示されますので，エラーが出た場合はそれを参照することで，だいたいどの辺りでエラーになったのかがわかります。以下のような感じです。

```
モーション0をコンパイル中.d.d..d.dr...d.dr..m.d.d...d.d.End
モーション1をコンパイル中.d.d..d.d..d.d.m.d.r....d.d.End
初期出現bottomをコンパイル中s....m.r.Err!
rパラメータの指定が不正です。
```

以上を踏まえて，サンプルのskeleton.msbのモーションフィールドを読み下してみましょう。

m0=0:(d(0),d(0)),1:(d(0),d(r(-1,0))),2:(d(0),d(r(-1,0))),m(0:(d(0),d(0)):128,1:(d(0),d(1)):128)
1.モーション0です。
2.キャラクターナンバー0を速度を維持(加速度0)で表示。
3.キャラクターナンバー1をY方向の加速度-1から0の範囲でランダムに与えています。
4.キャラクターナンバー2を上と同様に表示。
5.モーションナンバー0へ確率1/2(128/256)，ナンバー1へ確率1/2で分岐します。
ナンバー1のときはY方向に1の加速度を与えています。
全体的に見ると，キャラクターナンバー0,1,2の順で表示しながらなんとなく上に向かって進もうとしていて，半分の確率で同じモーションをループ，半分の確率で次のモーションに分岐しています。

m1=2:(d(0),d(0)),2:(d(0),d(1)),2:(d(0),d(0)),m(0:(d(0),r(-2,-6)):8,1:(d(0),d(1)):248)
1.モーション1です。
2.キャラクターナンバー2を速度維持で表示。
3.キャラクターナンバー2をY方向に加速度1。
4.キャラクターナンバー2を速度維持で表示。
5.モーションナンバー0へ確率1/32(8/256)，Y方向には-2から-6の範囲で速度を，モーションナンバー1へ確率31/32，Y方向に加速度1で分岐。
全体的に見ると，キャラクター自体はアニメーションせず，なんとなく落ちていっている感じ。かなりの確率でループするが，思い出したようにモーション0に分岐し，そのときには上向きの初速が与えられています。

bottom=s(-2,2,-10,6),m(0:(r(-1,1),r(-2,-6)))
1.画面下端からの出現です。
2.キャラクターの速度の範囲は右向き最大2，左向きも2，上向き最大10，下向き6。
3.モーションナンバー0へ連結。X方向は-1から1の範囲で，Y方向は-2から-6の範囲で　初速が与えられています。
全体的に見ると，ま，要するに下から現れて上に向かうんだよって感じです。

以上のようなデータが，実際にどのような動きになるかは，skeleton.msdをシェルに放り込んで，実際に自分の目で確認してください。

【拡張子について】
ここで拡張子を整理しましょう。

.BGM　BackgroundMascotShell設定ファイル(BackGroundMascot)
.MSB　MascotBuilderソース(MaScotBuilder)
.MSC　マスコットネイティブデータ(MaSCot)
.MSD　マスコットデータ(MaScotData)
ほとんど冗談みたいですが，これらを混同しないようにしてください。

【使用上の注意】
MascotBuilderはまだ開発途中ですので，どんなバグが潜んでいるかわかりません。各自の責任の下に使用してください。また，完成したデータの配布時には，十分な動作チェックを行ってください。

【試用上の注意】
MascotBuilderはまだ開発途中ですので，再配布は絶対厳禁です。また，不具合を発見した場合は，速やかに作者まで報告してください。

【仕様上の注意】
モーションデータには，スペース，タブ，改行などを絶対に入れないでください。文字数は1023文字までです。また，存在しないキャラクターナンバーを指定しても，エラーは出ません。

【その他】
マスコットデータは，作者のホームページのひとつ Mascot Collections にて一般公開されています。もしこのページに掲載してもいいというデータがありましたら，作者までご連絡ください。

Mascot Collections
http://www.hinix.com/kisawo/mascot/

【著作】
MascotBuilder version 0.20

E-mail:kisawo@hinix.com
URL:http://www.hinix.com/kisawo/

Game Fatale
私を変えたゲーム

スペースチェイサー

清水和人　Shimizu Kazuto

図1　私の記憶の中のスペースチェイサーはこんな感じ

私はよく思い出す。遠い昔のゲームセンターのずらりと並んだテーブルゲームを。喫茶店がことごとくブロック崩しやスペースインベーダーを並べていた時代を。ボスコニアン，ラリーX，パックマン，懐かしくも怪しい時代の寵児たちを。インベーダー，ブロック崩し，ギャラガ，ミサイルコマンド，セロハンで色づけされたインベーダー，名古屋撃ち，壁に貼られたハイスコアを。パチンコの危機といわれた時代を。あの時代はなんだったのか。

私はその時代を思い返すとき，必ずあるゲームを思い出す。自分を変えたあのゲーム，マイナーで地味なあのゲーム，初恋のような切ない思い出。それは確かタイトーのゲームで「スペースチェイサー」という名前だった。その画面は確か右図のような感じだったと記憶している。

なにかしみじみとした書き出しになってしまったけど，下側からスタートする三角形のフラスコのような宇宙船で，上側からスタートするやや細長いミサイルの追撃を振り切り，すべてのドットを消すと1面クリアという，「ドット消し」の見本のような作品だ。ちなみに確かナムコのパックマンと同時期のタイトーの作品だというから，ちょっとキモいかも。

ボタンを押すと燃料があればスピードが上がるという，緊急噴射機能もあったなあ。ボーナスのフルーツが出てくるわけでもなく，いまから見るといかにも初期の素朴なゲームといった感じだ。－しかしなにも持たない素朴なゲームだからこそ皇帝（おう，と読む？）を打つこともあるのだ？－

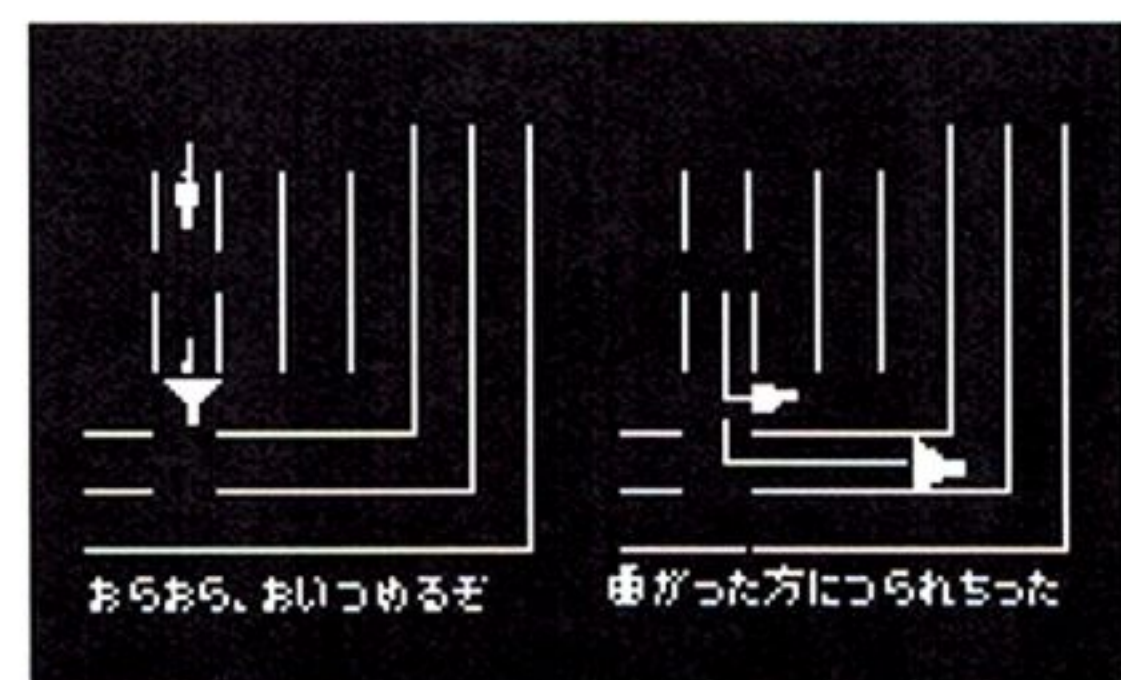
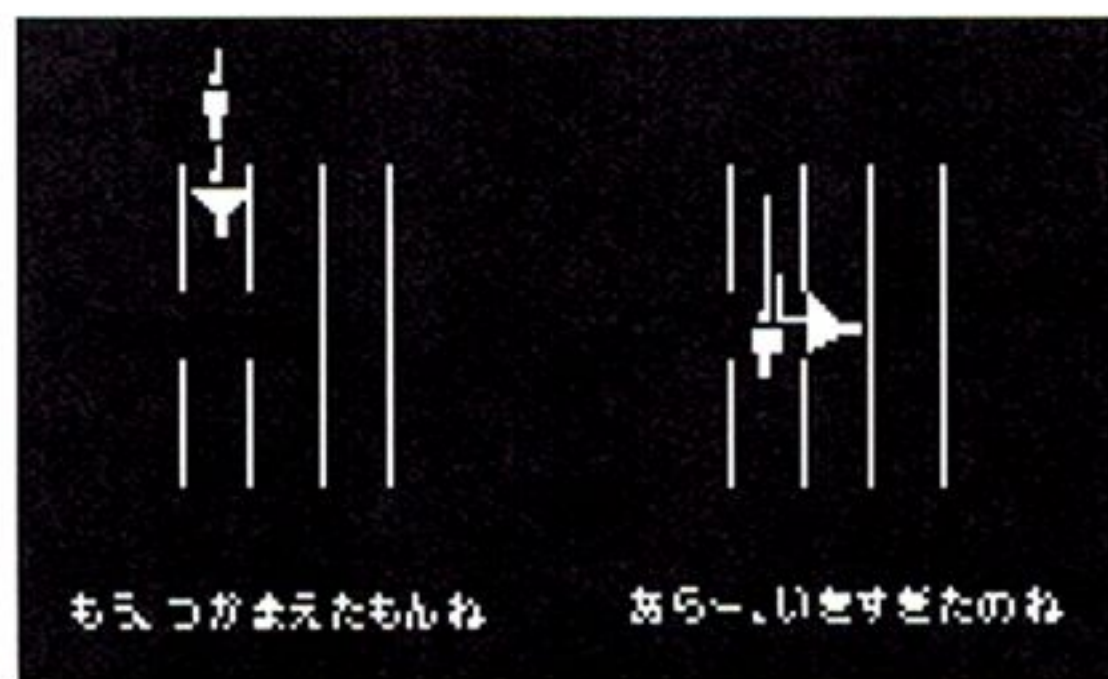
いつも思うんだけどゲームだけは100円の相場がほとんど変わっていない。ゲーム本体にはどんどんお金がかかってきているから（石は安くなってるだろうけど），実質どんどん値下げしているようなもんだ。ほんとに儲かるの？　なんか昔のゲームセンターはもっと地味なのに流行っていたような気がする。ゲーム機の進化が文化のバランスを変えてしまったのだろうか。

話は戻ってこの「スペースチェイサー」，すなわち「宇宙の追跡者」は，いとやむごとなききわにはあらぬが，そこそこのプレイヤーがついた中堅どころのゲームといっても許してもらえるだろう。

このゲームにハマる原動力となるのは理不尽な悔しさである。ドットを次々に消していき，残りがわずかになると，突然ミサイルのスピードが速くなり，緊急噴射ボタンを使っても逃げ切れなくなる。最後はいかにもおカマを掘られたという感じで「ドカーン」とやられてしまうのだ。これが非常に恥ずかしく，その悔しさでついついもう1回となるわけである。それも，あと1ドットか2ドットのところで容赦なくやられたりすると，かっとしてまた千円札の両替に走ってしまうのだ。初心者は1面，2面あたりでやられるので，もの凄いペースでお金を払い続けることになる。

少しやり込んでくると1面・2面あたりはパターンどおりに動くことでクリアできるようになってくる。ミサイルの追撃アルゴリズムが少しずつ肌で感じられるようになってくるのだ。

このゲームの第1のポイントはミサイルが後ろから接近しないようにすることだ。あまり近いとちょっとした操作ミスですぐ「ドカーン」となるからだ。だいたいの場合は少しずつ後ろから差を詰められ，ついには追いつかれてしまうことが多い。そこで後ろについたミサイルを振り払う技が必要となってくる。しかも最後にミサイルが加速するからできるだけ燃料はとっておきたい。燃料を使わずに振り切れないものか。

図2はこのように後ろにミサイルがついたときに，その追撃を惑わす方法として非常に有効である。ミサイルを後ろに引き連れ，中央の道を縦に移動する。そして90度曲がって外から2番目の周回路に入るのである。これをやるとミサイルの追跡アルゴリズムが破綻し，もうひとつ内側のところで同じ方向に曲がってしまうのだ。これで逆戻りするとかなり相手を引き離すことに成功である。これを覚えると後ろにつかれてもあせってミスしたりしなくなってくる。この動きでミサイルを見事にだますと，気持ちいいんだな，またこれが。

これでしばらくはドット消しに専念でき，また後ろにつかれたら同じ技を繰り返せばよいのだ。緊急噴射に頼っていたのがうそのように楽にプレイできるようになる。これがプレイヤーとしてのレベル1である。

ところがこの境地に達しても，もっと近くにべったりと追尾されるとお手上げである。緊急噴射で，ある程度離してから上の技に入るのだが，それでは最後の激ミサイルモードのときに燃料が残らなくなるのだ。そこでもっと便利な次の技を習得する必要が出てくる。どの技も追尾アルゴリズムの盲点（いや，手加減か）を突くのだが，今度のはちょと気づきにくい盲点である。

図3のようにかなり接近した状態でやはり中央の道を縦に移動する。そして画面のど真ん中に来たときに，くいっと横に入るのである。そうするとミサイルは「おっととと」というわけで曲がらずにいってしまう。これでいまきた方向に戻っていけば，かなり離すことができるのだ。このぐらい接近されるとかなりのピンチであり，あわてて逃げることに専念すると外周の距離のある部分に活路を見いだしがちであるが，実はこの危険とも思える方向転換が正解なのである。

以上の2つの方法を習得すると基本的にはこのゲームは終わりで，あとは実戦で精度を高めていくだけである。何面か進むとパターンも同じになってくるため，当時は長時間（半永久的？）にわたってプレイできる人が（少なくとも私の知っているゲームセンターでは）続出したのである。

おらおら，おいつめるぞ　曲がった方につられちった

図2

もう，つかまえたもんね　あら～，いきすぎたのね

図3

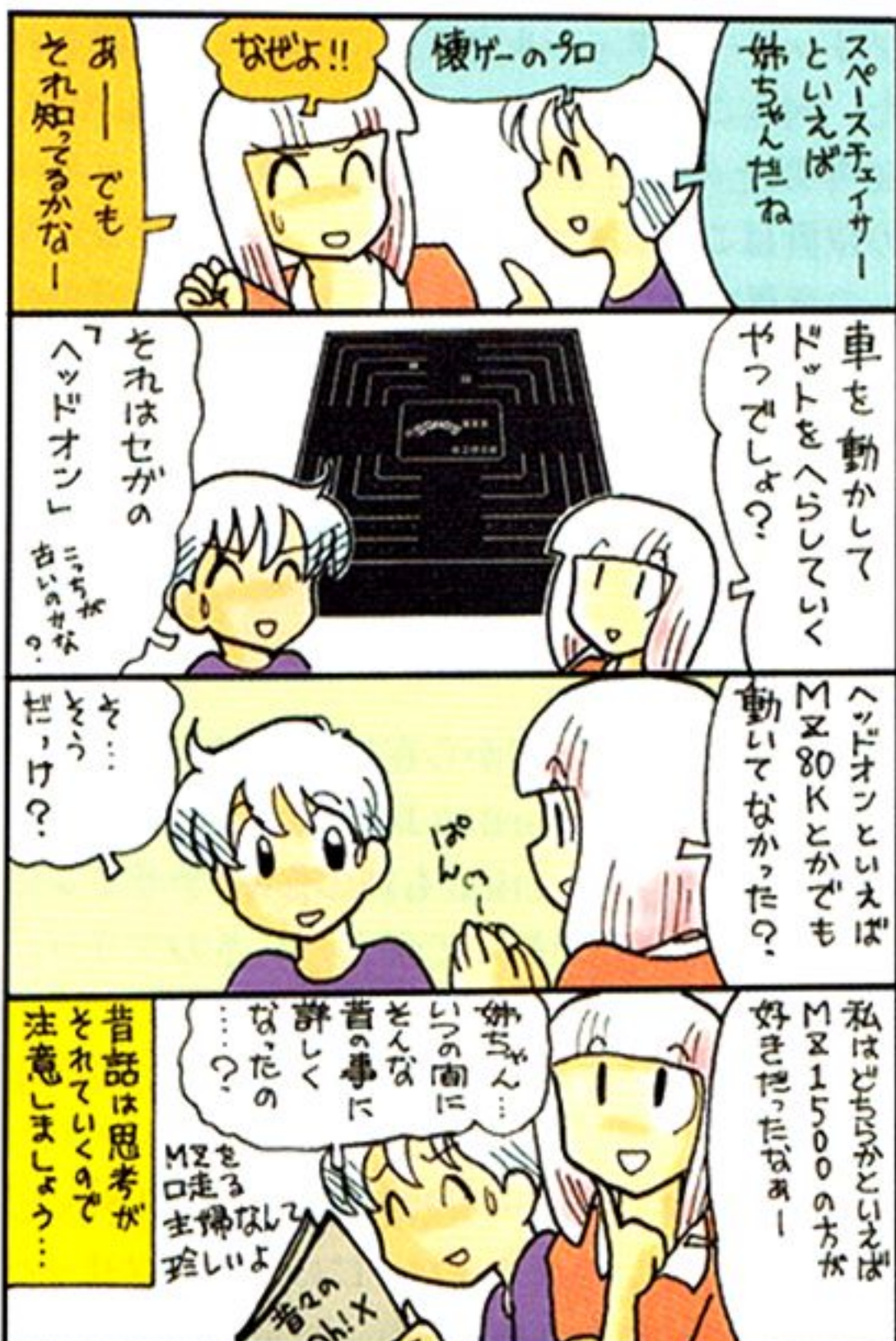
慣れすぎの法則

しかしさ，理論的にはそうかもしんないけど実際に連続して何面も何十面もクリアしていくには，やはりかなりの辛抱が必要なんだよね。やっぱそこだよね，ゲームっつうのは。スペースチェイサーの基本は安全にクリアできる動きのパターンを探すことから始まるさ。ある程度ほぞを噛むとパターンを見つけることができるさ。でもパターンを確実に身につけるには「慣れすぎの法則」を乗り越えなきゃね。

「慣れすぎの法則」っていうのは，せっかく見つけた安全パターンが，操作が習熟しすぎてずれちゃうことなんだよね。この場合は位置関係が微妙に異なることで，ミサイルがいままでと違うパターンの動きをしたりして，こっちも混乱して「ドカーン」とやられちゃうことがあるんだよ。こういうのに陥ったなら一度反省して，どこかで機体を壁に向けて一瞬休むとかしてタイミングを計る必要があるんだよね。そうこうしているうちに投資金額もかさんでくるんだ。小遣い少ないくせになかなか帰れなくなる「ゲームセンターシンドローム」の始まりだ。

●ゲームセンターシンドロームのいろいろ

・最初のプレイはすいてる台で（恥ずかちいもん）
・無残にやられると帰れない（ちっくしょうめい）
・うまくいっても帰れない（もっかい確かめよう）
・うまい人の技を横目で見る（ああそうやるのか）
・うまい人がいる間はできない（恥ずかしいって）
・ガキは嫌いだよ（見てんじゃねえよ）
・さっきまではできてたのに（ああ帰れない）
・カップルでくるんじゃねーよ（昔はね）
・並ばない（礼儀，前の人ゆっくりやってよね）

あせったり疲れたりしてくると，今度は曲がるべきところで曲がれなかったり，角で止まって追いつかれ「あーっ」「ドカーン」。これも連続クリアの難しさのひとつ。結局，習熟とアイデア，これがゲームというものの本質ですぜ，だんな。理論だけでないところに感情があり感動もある，実に人間的な文化ではありませんか。

スペースチェイサーの場合，パターンの発見と習熟というゲームの本質が，実に素直に表現されている。このようなゲームの場合，端から見ているより実際にプレイしている人の心理は，はるかに純粋で複雑で人間的なのである。必死でプレイするなかで「あれっ？　いまの動きは？」と，たまたま後ろについていたミサイルが機体のある動きについてこなかったことに気づけば，追跡アルゴリズムの穴をモノにできるのである。

気づく→試す→習熟，この喜び，慣れる→失敗→学習，この喜び，人のプレイを盗むより自分で開拓することのなんという楽しみか。

このスペースチェイサー，実は得点が10万点だったか，100万点だったかどちらか忘れたが，ある点以上は0に戻ってしまったと記憶している。ハイスコアの出しがいのないゲームだが，当時のゲームセンターはハイスコアを紙に書いて張り出す傾向にあった。このゲームに思い入れの強かった私は，ある日スコアにならないハイスコアを出すことを計画した。池袋・高田馬場・新宿・渋谷といった繁華街のゲームセンターは常連が多く，また人そのものも多いので，長時間プレイするには不適切だった。そこで場所は郊外に出て吉祥寺のゲームセンターに決めた。時間的に5～6時間と踏んでいたので，前日はよく寝て鋭気を養い，万全の態勢で臨んだのである。

長時間連続クリアの心得

・安全プレイを第一に，危なくなったら噴射ボタンを使う
・落ち着いて必要ならもう1回ミサイルをまくパターンを使う
・得点が一巡したらマッチ棒を1本置く（得点がわからなくなってしまうので）
・キリのいい100万点（だったか1000万点だったか，マッチ棒10本だったか）で終わりにする
・トイレに行っておく
・腹ごしらえをしておく
・飲みものを用意しておく

長時間プレイの心がけのいちばんは安全プレイである。サッカーオリンピックやワールドカップと同じである。なにせきりの悪いところでミスして終わってしまうと，大きな精神的ショックを受けるので，一見確実になってきた技でさえ，油断は禁物である。朝10時頃から始めたろうか，終了の予想時間は午後2時ごろ，そのあとゆっくり遅い昼食をとるつもりである。マッチ箱を持ってある台を選んで座って，私はおもむろにプレイを始めた。

それは長い時間であった。前日よく寝ていなかったら途中で集中力が途切れていたかもしれない。果てしなく続くパターンプレイ，少しのミスとリカバリ，時々店の人が見にくる以外は変化のない空間。一度か二度くらいはミスったかもしれない。とにかく昼も過ぎ，3時頃ようやく私はその日の目標を達成した。もうこのゲームをやることは二度とないだろうと思ったことを覚えている。そう，気分は「卒業」である。

もうあれから？？年近くの歳月が過ぎ去ったいま，ゲームに関していちばんに思い出すのはこの日のことである。いまそのときの気持ちを理解することはできない。どう考えてもぱっとしないこのイベントが私にとって大きな意味を持っていたのはなぜか。当時の私はなにを探していたのか……。

歴史にとってスペースチェイサーとはなんだったのか？　X68000とはなんだったのか。計算機・ソフトウェア・ネットワークという3つの技術によって発展してきた最近のコンピュータ文明，それらはいつまで文明の原動力であり続けるのか。Oh!Xはシャープとは別の文化として2001年（21世紀）を迎えるのか。

私的文明の潮流予想表（ハルマゲドンなし）

項目	100年後	1000年後
テレビ	かなり便利に	なにかに進化
Macintosh	ない	ない
Windows	ない	ない
UNIX (Linux)	ない	ない
計算機メーカー	もう文明の中心ではない	全部なくなっている
ゲームセンター	なにか別の形で似たようなもの	全然新しいもの
コンピュータゲーム	凄くなってる	物凄い

私的文明の潮流予想表によると100年後にはいまのメジャーなモノは本流ではなくなっている。もちろんX68000とかスペースチェイサーという単語は意味すら誰も理解できなくなっているに違いない。メジャーなWindowsですら，多分「初期の計算機」と書かれて博物館に並んでいる程度だろう。「あらまきれいとよくみたら」とサンヨーのカラーテレビが発売されてからまだそんなに経っていないし，ましてやパーソナルコンピュータなんてようやく幼稚園の年少組といった年齢である。それでもそのわずかな時間の中で当事者の我々から見ると大きな変貌を遂げているのである。その大きな変化も100年，1000年という歴史のなかでは実に小さなイベントにすぎないのか。

そう，歴史の中では「計算機の発展」「ネットワークの浸透」「ソフトウエア技術の発展」という本質のみが真理として残るのである。「ゲームは人間の感情の文明」「ゲームがコンピュータを進化させる」などもそのような本質の流れとしてよいかもしれない。そのような「真理」的なものが文明の緩やかな発展として残っていくのである。

しかしいくら「歴史は本質」とうそぶいてみたところで，歴史は実体としてのイベントこそがそのすべてであることに変わりはない。つまりスペースチェイサーこそが歴史の実体であり，名前としては残らなくても未来の文明に影響を与える因子なのである。そしてそこには自分自身がいる，感情がある，人間がある。いくらハイテクとはいえ，計算機の世界でも最終的には個人個人の思い入れこそが重要なのだあ。だあーーーーーーー。

Flipper Pinball Stories #3
フリッパーの変遷

市川幹人　Ichikawa Mikito

現在のタイプのピンボール台の主役ともいえる「フリッパー」について見ていこう。単にボールを跳ね飛ばす仕掛けだった時代からボールを制御する仕掛けへの変遷を追ってみる。

Pre Flipper ?

いまだ電気を使うゲーム機の存在しなかった頃，すでに「プレイヤーの意思で操れる重力に逆らえる仕掛け」は存在した。Baffle Ball登場から約1年，〝ついに〟というべきか〝やはり〟というべきか，フィールドの中にプレイヤーの意思で操作可能な仕掛けが登場する。Hercules NoveltyのDouble Shuffleだ。Hercules Noveltyの社長，Charles Chizenwer氏はピンボールメカにおける新たな原理ともいえる発見をした。〝軸を持った支点によりものを動かす〟ことだ。いまなら子供でも考えそうなことだが，1932年当時このアイデアはまだピンボールに使われていなかった。

Double Shuffleは半年にわたりノースカロライナ州でテストを行い，満を持して1932年9月に登場した。Buffle BallやBally Whooが話題になって間もない時期に登場しただけにたいへんな注目を集めた。しかし，このゲームはGottliebの特許を侵害しており，これが原因で成功とはならなかった。また，この大きな〝フリッパー（？）〟があることでフィールドの玉の流れは極端に制限され，発展性のないアイデアだったのか，翌年10月に登場したペイアウト式ピンボールが大ブームになったためか，このあとこのタイプのゲームは登場しなかった。

フリッパー登場
1947年10月　Gottlieb Humpty Dumpty

ペイアウト式のピンボールのため，多くの州でピンボールが違法となっていたなか，GottliebのHarry Mabs氏はフリッパーを発明する。

彼は〝ベースボールゲームのバットを何度も打ち返すことはできないのか？〟と考えていた。そして発想したのがフリッパーだ。〝何度も打ち返せないのか？〟という発想からきているので，最初は「狙う」というよりは「打ち返す」または「打ち上げる」という印象が強く，ボールを〝コントロールする〟という発想は弱かったようだ。

名前からして〝Flipper Bumper〟(Bumper＝跳ね飛ばすもの）と呼ばれていた。現在のピンボールプレイヤーのなかにもフリッパー登場から1960年代後半までのピンボールを好まないプレイヤーも数多い。この頃のフリッパーに対して現在のフリッパーが〝狙う〟役割が強いからだろう。

〝狙う〟，〝ボールをコントロールする〟という発想が弱かったことを裏づける理由のひとつに，Humpty Dumpty以降しばらくはホールド不可能な（フリッパーボタンを押しっぱなしにしてもフリッパーは一瞬だけ上がりすぐに下がる）作品や，フリッパーボタンは2つあるものの，両方のフリッパーが同時にしか動かない作品は少なくなかった。

フリッパー登場の影響は大きく，ピンボール自体が「フリッパー」と呼ばれるようになり，フリッパーなしのピンボール（いわゆるスマートボールやコリントゲーム，認めたくはないがパチンコ）とは決定的に違うゲームへと進化することになる。

Humpty Dumptyからわずか3カ月，GencoからTriple Actionが発表される。この作品ではフリッパーはフィールド下部に左右対に配置された。向きこそ現在のピンボールとは違うものの，50年以上の年月が経過した現在でもフリッパーの位置はこのままである。フリッパーはプレイヤーの意思で操作可能な重力に逆らえる仕掛けなので，フィールド下部に配置したことで〝重量に逆らえる〟特徴がよりはっきりした。

現行の長さの3インチフリッパー登場
1968年8月　Williams　Hayburners II

Williamsの草創期から在籍し，現在もデザインを行うSteve Kordek氏（なんと現在89歳，GencoのTriple Actionも彼の作品）デザインのHayburners IIで初めて現在の長さのフリッパーが登場した。

それまでのフリッパーよりも格段に〝狙える〟ようになった，長くなったことでフリッパーのパワーも増し，フリッパーの役割がさらに強調された。現在ではメインのフリッパーには，短いフリッパーはまったく使われなくなり，以前の短いフリッ

Base ballゲーム
フリッパーピンボールの前身のひとつ。野球盤の大型版？　いまでも，ときおり新作が発表されるジャンル。野球好きな人が2人で遊ぶと意外なほどに盛り上がれる

Baffle Ball

Double Shuffle

Double Shuffleには手動式フリッパー（？）とも感じ取れるバーが8組存在する。手前にあるレバーで動かすことができる。機敏に動かすゲーム性ではなく〝打つ〟というよりは〝跳ね上げる〟感じだった。この程度の装置を安定動作させるのにもこの当時は半年もの時間を要した。〝The SENSATION of 1932〟と書いてあるのが象徴的！　下部左側に〝All Skill〟と書いてあるのもこの手動式フリッパーが斬新であることをアピールしている。Junior Model（約41cm×81cm），とSenior Model（約56cm×115cm）があり値段が違う。この当時は2タイプのモデルがあることと，足が別売りなのは珍しくなかった。フィールド下部を見てほしい。Buffle Ballと同じである。これが結果的には特許の侵害になってしまった

Humpty Dumpty
フリッパーの存在感をアピールするためか6枚もついている。配置としては「重力に逆らう」「ボールをコントロールする」フリッパーの2大特徴をまったく生かされない配置だったが，それでも自分の意思で動かせるフィールド内の装置のついたこのピンボールに人々は夢中になった

Triple Action
Humpty Dumpty登場からわずか3カ月。現在のフリッパー位置にフリッパーが配置される。デザイナーのSteve Kordekはその後も数々の仕掛けを考案したが，その才能がすでに垣間見られる作品だ

パーを〝ショートフリッパー〟と呼ぶように区別されたほどだ。

この試みはすぐにほかのメーカーにも影響を与え1969年にBallyはBallyhooで，Gottlieは1970年にPlayballで，この長いフリッパーを採用。この影響を受けすぎたのかどうかは不明だが，Chicago Coinは5インチもあるフリッパーを搭載したBig Flipperを発表した。

また，ボールのホールドがやさしくなるため，初心者プレイヤーにもボールをコントロールすることが可能となった。フリッパーレーンと絡み，「フリッパーレーンからのボールを隣のフリッパーに渡してホールド」といった動作はショートフリッパーでは大半のプレイヤーには実用的なテクニックではなかったが，フリッパーが長くなって以来，多くの人に新たなゲームの組み立てを可能とさせた。

この一見たいしたことのない発想は，このあとのピンボールを大きく変えた。また，フリッパーレーンはロングフリッパーにより一挙に標準的な仕掛けになったといえる。

また，1986年に登場したWilliamsのHi-speed以降のマルチボール＆ジャックポットの台ではマルチボール中にひとつのボールを残してすべてのボールをホールドする作戦は有効な手のひとつだが，これもロングフリッパーの存在を不動とする原因になったと筆者は考えている。

余談になるが筆者はショートフリッパーの新作がまったく作られなくなったことは非常に残念だと思っている。微妙なタッチでボールを捌くゲームよりも派手な展開のゲームのほうが多くのプレイヤーに受け入れられることは容易に理解できるが，ショートフリッパーの台では1ショット1ショットがゲーム展開に非常に大きな影響を与え，趣のある展開でボールを捌く楽しみはむしろ現在のピンボールにはない味わいがある。現在のフリッパーピンボールとは別のジャンルといえるほど違った印象を受け，現代のフリッパーピンボールとの接点は一見非常に薄く感じられる。

プレイヤーとしてフリッパーピンボールが上手くなりたい人は一度プレイされることをすすめる。1ショット1ショットに集中できるようになり，現在のフリッパーピンボールのプレイスタイルにも大きな影響が出るはずだ。

今回は，ガラスで遮られたフィールドの中でプレイヤーの手として働くフリッパーの概要について記した。フリッパー登場の頃からフリッパーボタンは台の側部に存在し左右対に存在する。フリッパー登場から50年以上が経過した現在，今後のピンボールに大きな影響を与える，プレイヤーの意思でコントロールできる装置の登場に期待しつつ，筆者は毎夜，ピンボールの下でピンボールの夢を見続ける。

次回は1980年代に登場したソリッドステートフリッパーについて記す予定だ。

協力
State Machine (http://pinballjapan.com/)
小糸光儀
ゲームコスモ蒲田店　石戸谷 克義

Fair Way
1954年Williams創立者Harry Williamsのデザイン。左右のフリッパーが同時に動き，ホールド不可能なピンボール。しかし，これはこれでひとつの作品として完成しており非常に楽しめる。"はまる"という意味では現在のピンボールよりもはまり，「もう1回！」となりコインを投入してしまう作品だ。

Reserve
1961年Williams，Steve Kordekのデザイン。短いフリッパー，フリッパーレーンのない頃の作品。現在のピンボールと比較してボールをホールドするのがたいへん難しい

Black Pyramid
現在の3インチフリッパー。フリッパーレーンとともにボールの扱いやすさは格段に上がった。1984年Bally製CPUボード搭載のピンボールだが，意図的にフィールド，ルールともに'70年代以前風にした作品。フリッパータッチも絶妙でアナログ的にフリッパーをコントロール可能。プレイしていると得点を上げることはそっちのけでボール捌きに夢中になってしまう

ポピュラス・ザ・ビギニング ネットワーク対戦マップの完全攻略

西川善司 Nishikawa Zenji

ブルフロッグが運営する対戦サーバー。URLは，http://www.populous.net/

ポピュラスの醍醐味は対戦にある。ワールドレベルでもトップクラスポピュラス3プレイヤー，西川善司の戦略を，2，3，4プレイヤー，すべてのパターンの対戦マップについて解説する。これを読んで，ぜひ対戦サイトで活用してみてほしい。

DWANGOのインターネット対応版であるIWANGO。こちらのURLは，http://www.dwango.co.jp/iwango/

現在，ポピュラス3に対応したインターネット上の対戦サイトは海外にひとつ，国内にひとつある。

海外にあるのは開発元のブルフロッグが運営する対戦サーバーで，URLは，http://www.populous.net/だ。ここでユーザー登録をすれば入会金無料，年会費無料，ポピュラス3のパッケージを持っていれば誰でもタダでプレイを楽しめる。海外サイトなので公用語が英語だが，対戦自体はアイコンをクリックして行くだけで始められるので英語が苦手な人でも問題はない。日本語版と英語圏版は表示言語が違うだけでゲームプログラムは同一なので，英語圏版との対戦はいうまでもなく可能だ。

国内にあるのはDWANGOのインターネット対応版であるIWANGOがある。こちらのURLは，http://www.dwango.co.jp/iwango/だ。このページでユーザー登録をし，ここで提供されているクライアントソフトを使ってサーバーにアクセスする。こちらも現在，入会金無料，年会費無料でフリーで楽しめる。ただし，IWANGOはポピュラス専用サイトではなく，その他のさまざまなゲームに対応した対戦サーバーであるため，必ずしもポピュラスの対戦相手が見つかるとは限らない。

対戦にあたって気をつけること

ポピュラス3の対戦を始めるにあたって，いくつか知っておきたいことがある。

●PINGは600以下が望ましい

対戦を始めようとすると，ゲーム開始前の画面でPINGという数値データが表示されているはずだ。このPINGの値は各プレイヤー間のデータ伝送遅延時間（ディレイ，ラグタイム）に相当する。この値が小さいほど，快適にプレイできるものと思ってほしい。

ポピュラス3の場合はだいたい600以下であればまあストレスなくプレイできるようだ。同一国内のプレイヤーであれば大抵は400以下になるはずだ。逆に海外のプレイヤーとはこの値が悪くなることが多いようだ。PING値が800以上のときはまはやプレイできるレベルの遅延時間ではないので，別のプレイヤーを探したほうがいいだろう。

● 相手のバージョン番号を確認する

日本語版ポピュラス3はバージョン1.01で出荷されており，一方，英語圏版は発売がいちばん早かったこともあってバージョン1.00（メニュー画面にはFINALと表記されている）で出荷されてしまっている。現在メーカーのサポートサイトにて英語圏バージョン専用のバージョン1.01へのアップデートプログラム（ftp://ftp.ea.com/pub/bullfrog/patches/populous/poptbpatchversion1_01.exe）が公開されており，日本語版とプレイするためには英語圏版はこれの組み込みが必要不可欠だ。

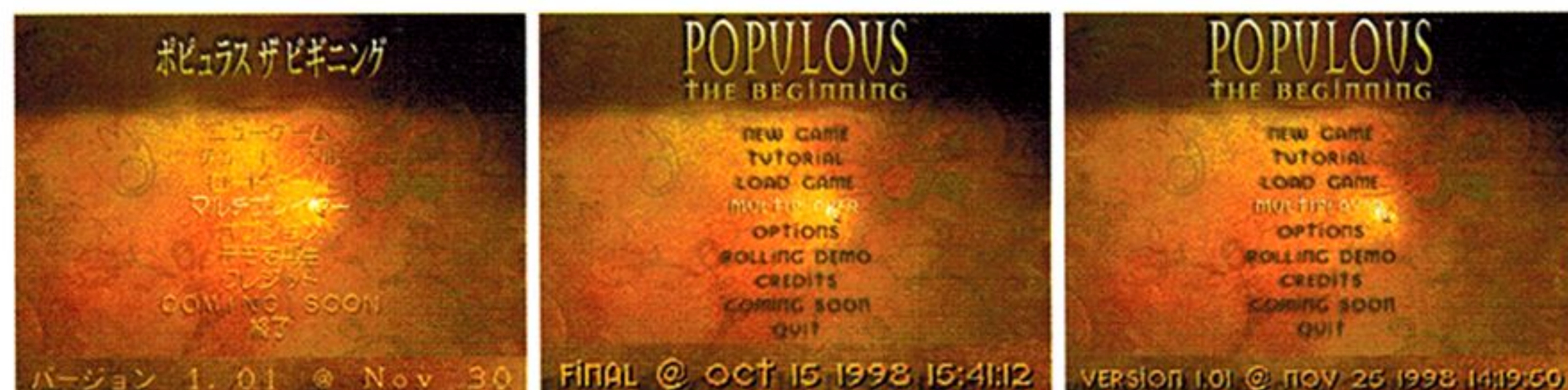
バージョンが違うとプレイ中エラーが頻発しそのうちゲームが強制終了されてしまうこともあるので，プレイ開始前にお互いのバージョンをチェックしたほうがいい。

●日本語版と英語圏版の非互換部分

日本語版と英語圏版はたとえバージョン番号を1.01に揃えたとしてもまともにプレイできないことがあり，対戦中，突然エラーが発生することがある。

エラーの発生確率が非常に高いのは，礼拝物へ礼拝を行ったときだ。

礼拝物，礼拝時のエラーは日本語版同士の対戦でもエラーが出ることがあるので事前に両プレイヤー間で礼拝物は祈らないように取り決めておくといいだろう。

なお，英語圏版同士の対戦ではこの問題が起こらないことから，日本語圏バージョンのほうに問題があると見られている。

左から日本語版，英語圏版出荷状態，アップデートプログラムを組み込んだあとの英語圏版

マルチプレイヤーゲームマップの攻略

●攻略マップの見方

各マップの攻略中に「ワールドビュー」（ゲーム中[Enter]キーを押して見られる惑星の全体図）の画面写真を数枚紹介している（必ず，画面上側が北になっている）。

戦いの舞台となる戦域を中心に紹介しているが，これ以外に，特殊スペルや建築物の知識，その他の奇跡を与えてくれる礼拝物の位置も示してある。

なお，それぞれ画面写真中に記載されている記号はそれぞれ以下の意味を持っている。

- B：プレイヤー1（青）ゲーム開始地点
- R：プレイヤー2（赤）ゲーム開始地点
- Y：プレイヤー3（黄）ゲーム開始地点
- G：プレイヤー4（緑）ゲーム開始地点
- 1～5（数字）：礼拝物。欄外にどういったものを与えてくれるかを記載している
- a，b，c～g（アルファベット小文字）：掲載されているワールドマップの連続性を表している。複数の画面に同一のアルファベットが記載されている場合は，その土地が互いに繋がっていることを表している

●方角について

文中に東西南北を使った方角表現が何度か出

てくるが，これはゲームの「マップ」や「ワールドビュー」における方角を表している。

たとえば，ゲーム中，テンキーの[0]を押すとゲーム画面の奥が北，手前が南，右が東，左が西……というふうに視点がリセットされる。

「マップ」を見るとマップ中に表示されているプレイヤーカラーの「○」の上に白い「△」マークが見えると思うが，この「△」が北を指し示している。

また，「ワールドビュー」のモードでは絶えず画面上方向が北になることも覚えておくと役に立つだろう。

2 プレイヤーマップの完全攻略

運命の丘
HILLS DIVIDE US

心がまえ

マップ名のとおり，丘が敵陣地との間に立ちふさがっている。

地形自体はお互いにほぼ対称で，同条件で戦えるマップだ。敵陣地とはかなり近い位置関係にあるため，敵との接触はかなり早い時期に起こるだろう。そういう意味では短期決戦に向いたマップということもできる。

野人コンバートを手早く

野人は比較的多いので勇士獲得にそれほど苦労はしないだろう。

しかしお互いの陣地が互いに近いせいもあり，ゲーム開始早々こちらの陣地にやってきて，こちら側の野人をアグレッシブにコンバートしてくるプレイヤーもいる。

伝道師で速攻をかけろ

互いの陣地が非常に近いため，速攻が非常に有効マップだ。

もっとも効果的なのは，伝道師をゲーム開始早々送り込む作戦だ。

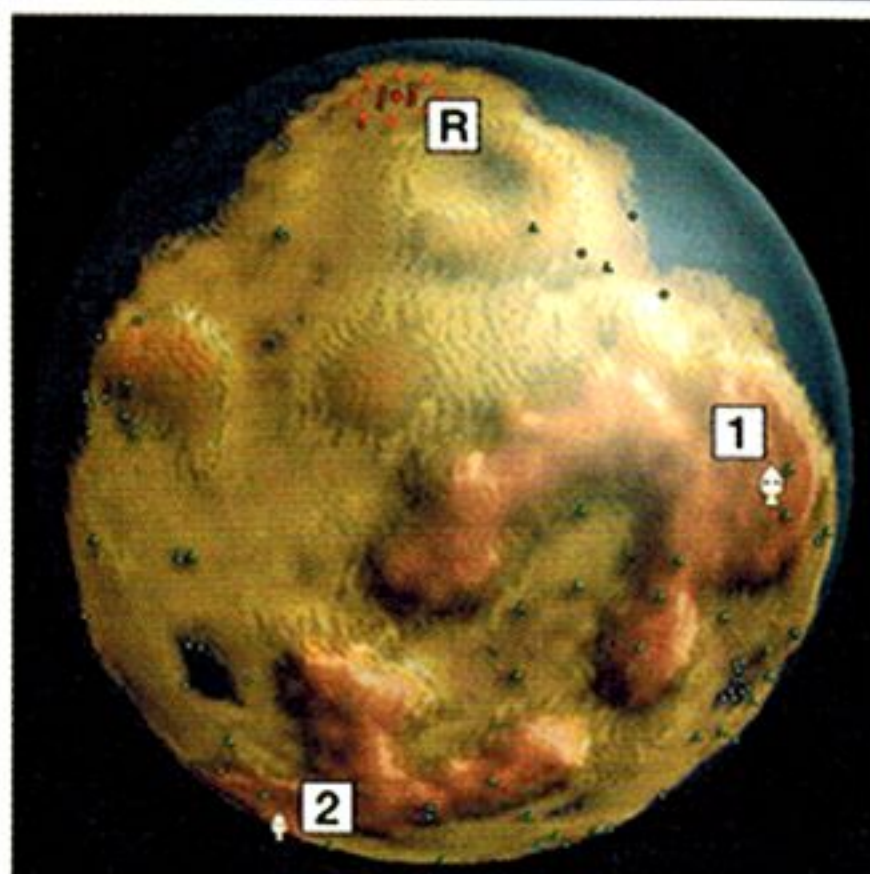
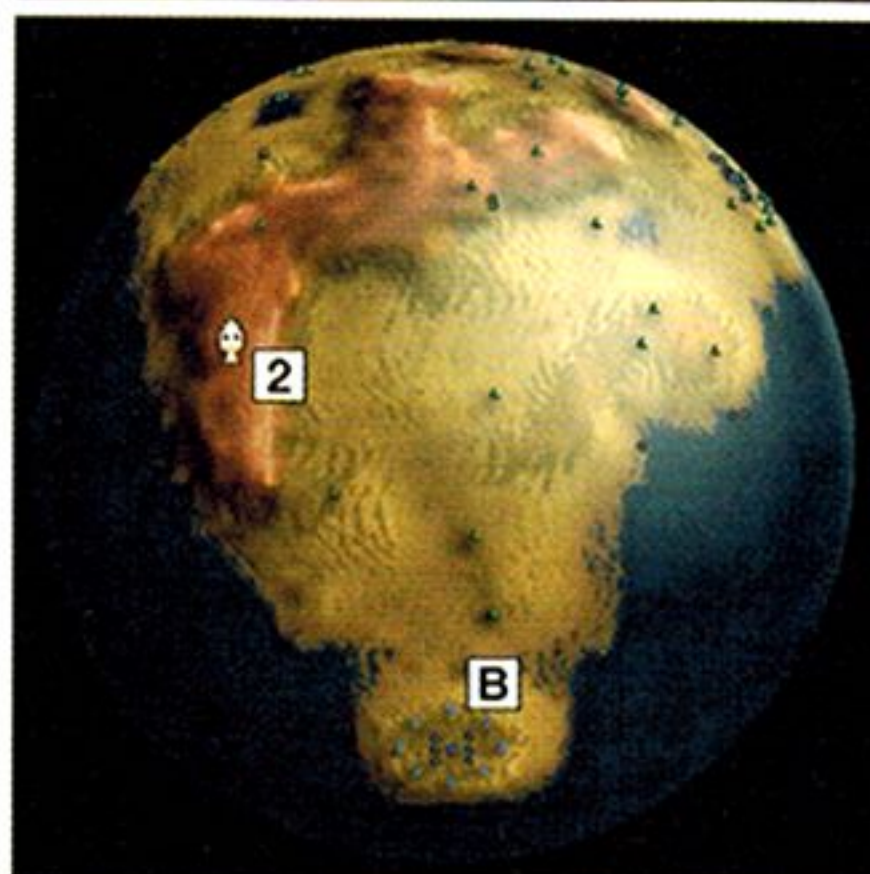
ゲーム開始直後にまず伝道師訓練小屋を建てて，すぐに数人の伝道師を仕立てるのだ。これを少しずつ断続的に送り込んでやる。相手プレイヤーがまだ伝道師訓練小屋を建てておらず，防衛ラインを作っていたり小屋を作っていたりするようであれば，作戦はほぼ間違いなく成功する。

敵伝道師がきたことを察知した相手プレイヤーは，自分も伝道師小屋を作ろうとしたり，あるいはすでにできている伝道師小屋に勇士を送り込んで伝道師で対抗しようとするだろう。これも先読みし，こちらの伝道師を敵陣の伝道師訓練小屋（建設中でもいい）周辺に送り込んでしまうのだ。

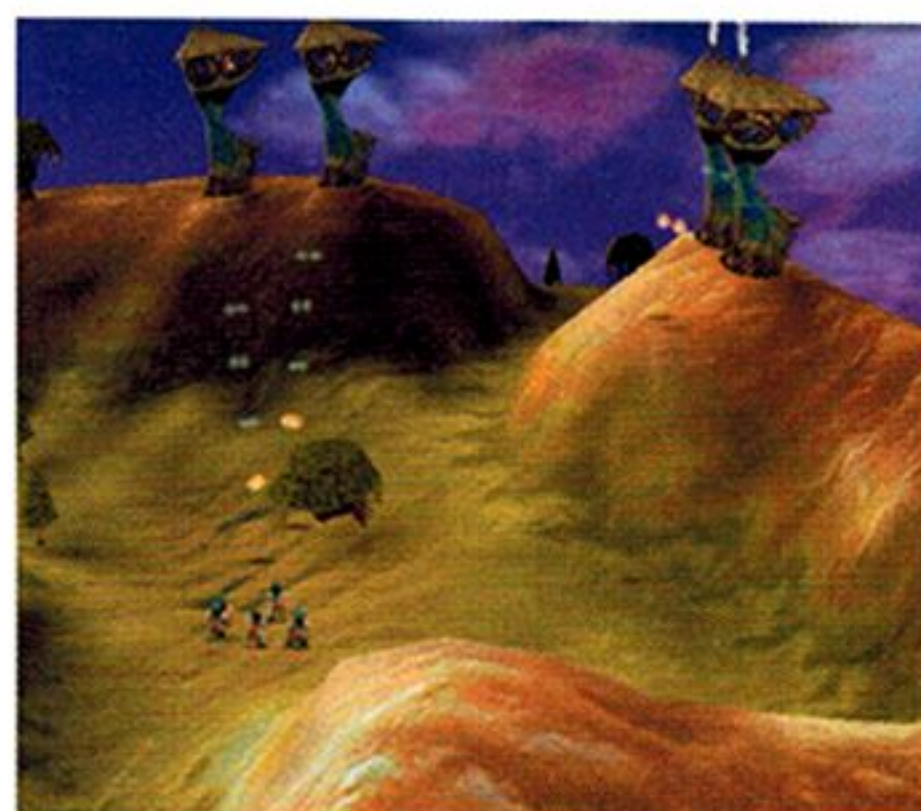
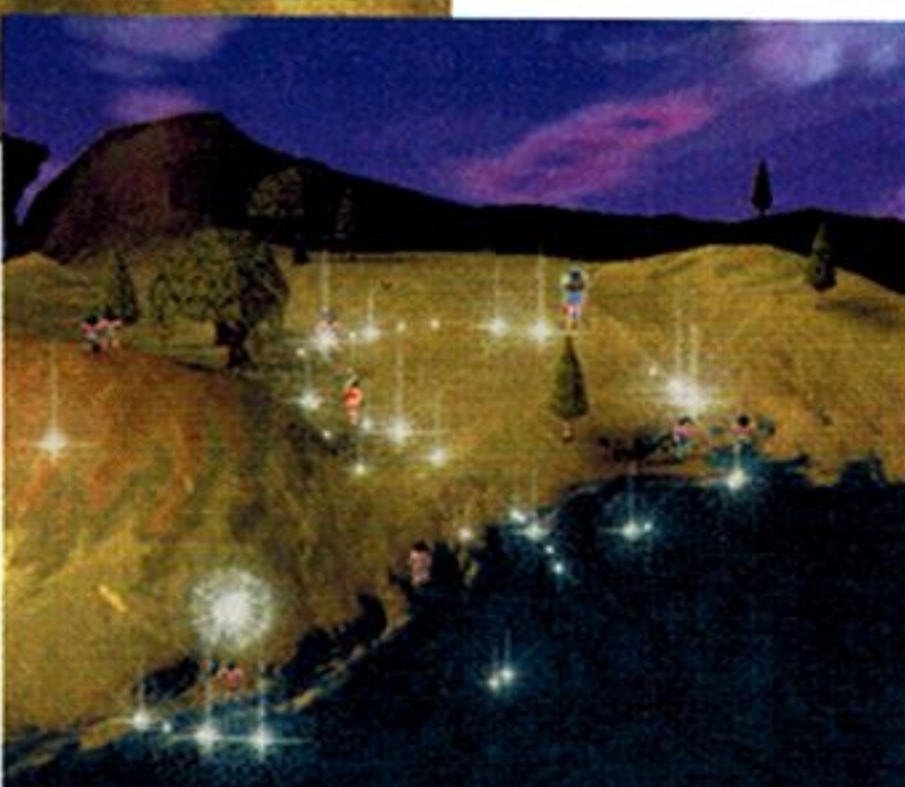
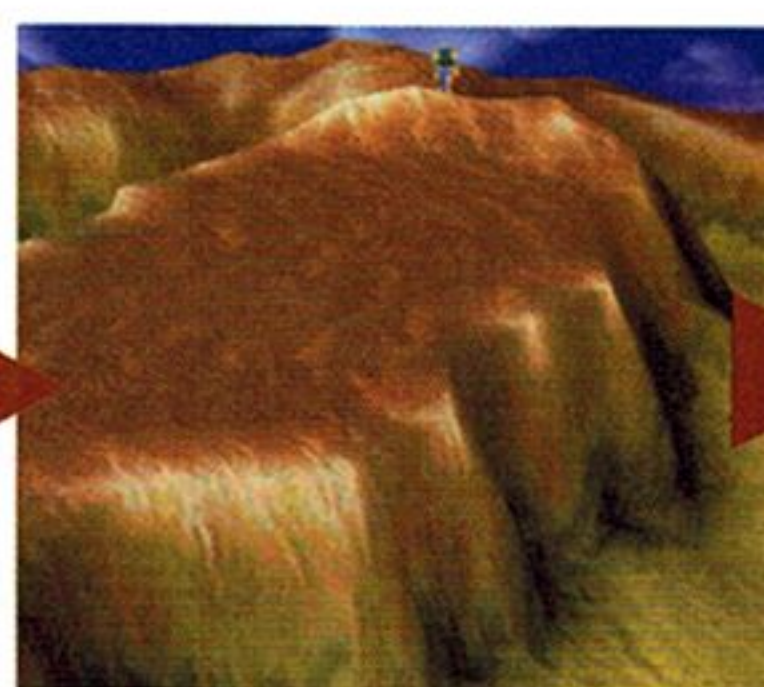
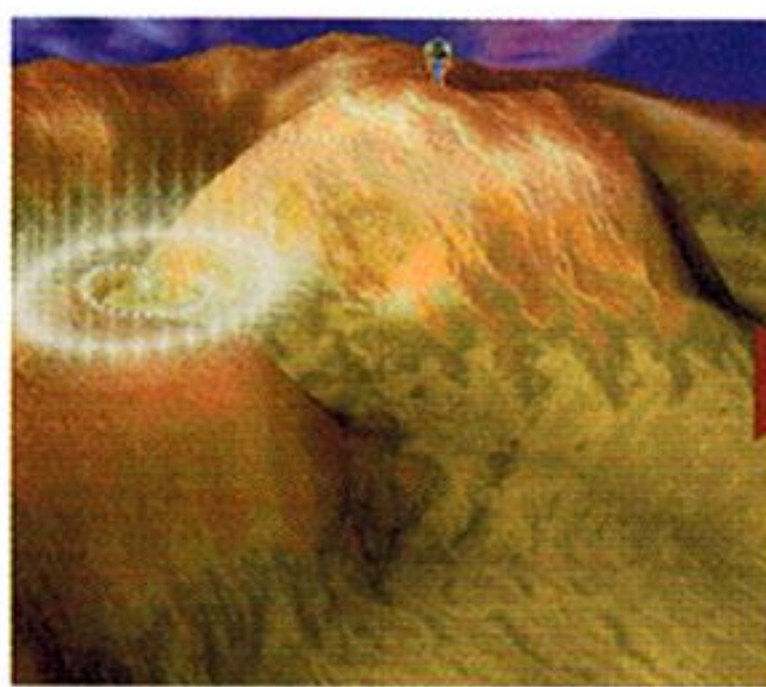
仮にこれら伝道師をなんとか退けたとしても，敵はだいぶ建設事業に遅れをとるはずだ。こちらは相手が混乱に陥っている間に，野人を勇士に変え，本拠地の建設をどんどん進めてしまおう。

最初から使えるスペル	
トルネード	ハチの大群
催眠	幽霊軍隊
ライトニング	コンバート
マジックシールド	ブラスト
透明	

最初から建てられる建物	
小屋	スパイ訓練小屋
ガードタワー	炎の戦士訓練小屋
戦士訓練小屋	ガードポスト
伝道師訓練小屋	

このレベルの概略図
1. 「浸食」スペル（×2）
2. 「土地平坦」スペル（×2）

ゲーム開始後まず作るべき派伝道師訓練小屋

先に敵陣へ伝道師を送り込めたほうがその後の戦いを有利に進められる

ゲーム序盤は野人の取り合い合戦になる

防衛ラインが完成するとここから侵攻するのはほとんど不可能にできる

丘の頂上に立ち，2発の「土地平坦」スペルを使えば，このマップ唯一の侵攻路を台地の壁で遮断できる

防衛ラインの形成

このマップは短期決戦になる場合が多いので防衛ラインを張るまでもなく勝負がつくことがほとんどだ。しかし，長期戦になる見通しがあるならば，防衛ラインを築いておいたほうがいい。

ベストポイントは互いの陣地の真ん中に立ちふさがっている丘の上だ。ここにガードタワーを建てて，ここに炎の戦士を住まわせると，ほぼ完璧な防衛ラインができあがる。

この防衛ラインが完成させてしまえば，「透明」スペル「マジックシールド」スペルなどを使わずに，この丘を通って侵入することはほとんど不可能にできるはずだ。

なんに使う?「浸食」スペルと「土地平坦」スペル

このレベルのストーンヘッドに祈ることで「浸食」スペルと「土地平坦」スペルが2発ずつ獲得できるが，これはいったいなんに使えばいいのか。

確かに使いどころは難しい。

あえて使うとすれば「浸食」スペルは丘の切り崩しを行うときに，「土地平坦」スペルは互いの陣地の往来に便利な唯一の谷間の道をふさぐ目的に使える。ただ，無理に多大な犠牲を払ってまでこのスペルを獲得する必要はあるまい。

台風の目

EYE OF THE STORM

最初から使えるスペル	
地震	土地隆起
土地平坦	透明
腐食	ハチの大群
トルネード	幽霊軍隊
催眠	コンバート
ライトニング	ブラスト

最初から建てられる建物	
ガードタワー	炎の戦士訓練小屋
小屋	ボート小屋
伝道師訓練小屋	ガードポスト
戦士訓練小屋	

心がまえ

互いの陣地は非常に近い位置関係にありながらも陸続きではない。よって戦闘の準備が整った場合は「土地隆起」スペルを使って敵陣地へ接続してから侵攻する必要がある。

もちろん，敵も同様の方法で侵攻してくるので，侵攻が予測される場所には防衛ラインを形成する必要があるだろう。

土地自体は広く木々も豊富なので建設はやりやすい。

まずは島の中心部へ移動する

ゲーム開始地点の「転生の地」周辺は土地がないので，全勇士を島の中心部へ移動させ，適当な場所に，まずはガードタワーを建てることから始めなければならない。ガードタワーを1本建ててからはその周囲に好きなように建築を進めていける。

侵攻ポイントはどこ?

お互いの陣地の島の東西両端が海で隔てられてはいるものの非常に近い位置関係にある。

見た目でピンとくると思うが，この東西の両端は「土地隆起」スペル一発で簡単に接続できてしまうのだ。

表向きはこの東西のどちらかが侵攻ポイントになり得るのでここに防衛ラインを形成するといいだろう。

ストーンヘッドがある離れ小島

互いの陣地の島のちょうど中央の位置にストーンヘッドが建っている離れ小島がある。

ここで得られるのは「浸食」スペルが3発だけなので，無理に取る必要はないが，ここに「土地隆起」スペルで陸を繋げ，さらに，もう一発敵陣へ「土地隆起」スペルを放つことでここを侵攻路にすることもできる。

島の東西にちゃんと防衛ラインを張っているプレイヤーも，ここを侵攻ポイントとして使えることを見逃している場合があるので，裏をかくには絶好の場所といえる。

もちろん，自分はここから侵攻されることを想定して，早期から防衛ラインを形成しておくこと。

裏をかく

プレイヤー1(青)は自軍「転生の地」の北東方向，プレイヤー2(赤)は自軍「転生の地」の南西方向を見てみよう。

なんか意味ありげな離れ小島(というか中州?)が目につくはずだ。

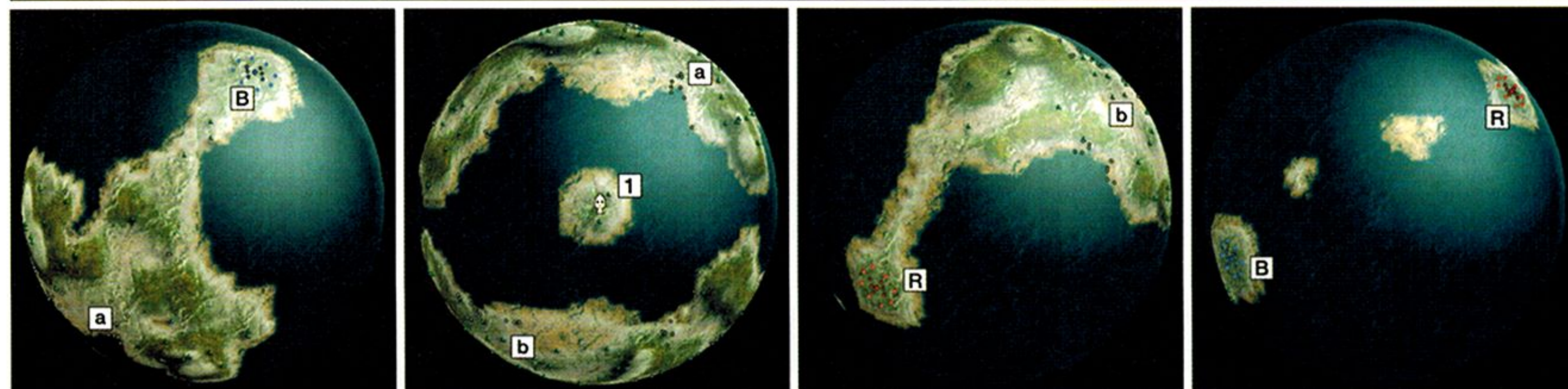
自軍「転生の地」からこの中州に向かって「土地隆起」スペルを放っていくと，敵「転生の地」と陸続きにすることができるのだ。

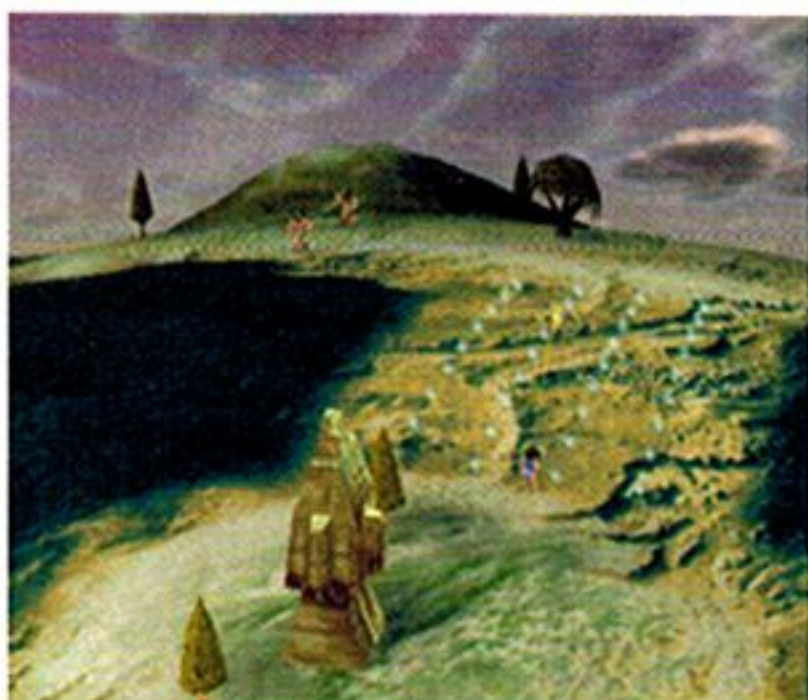
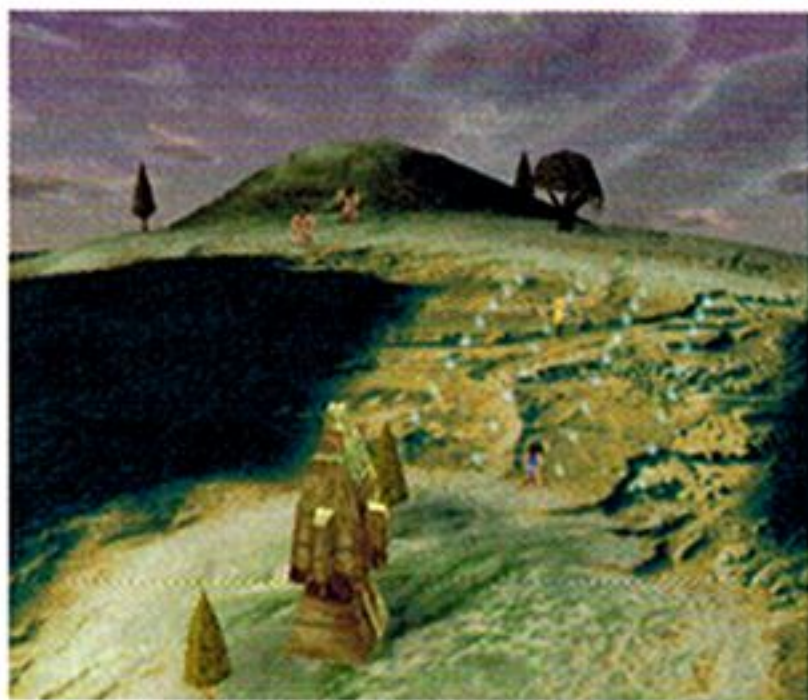
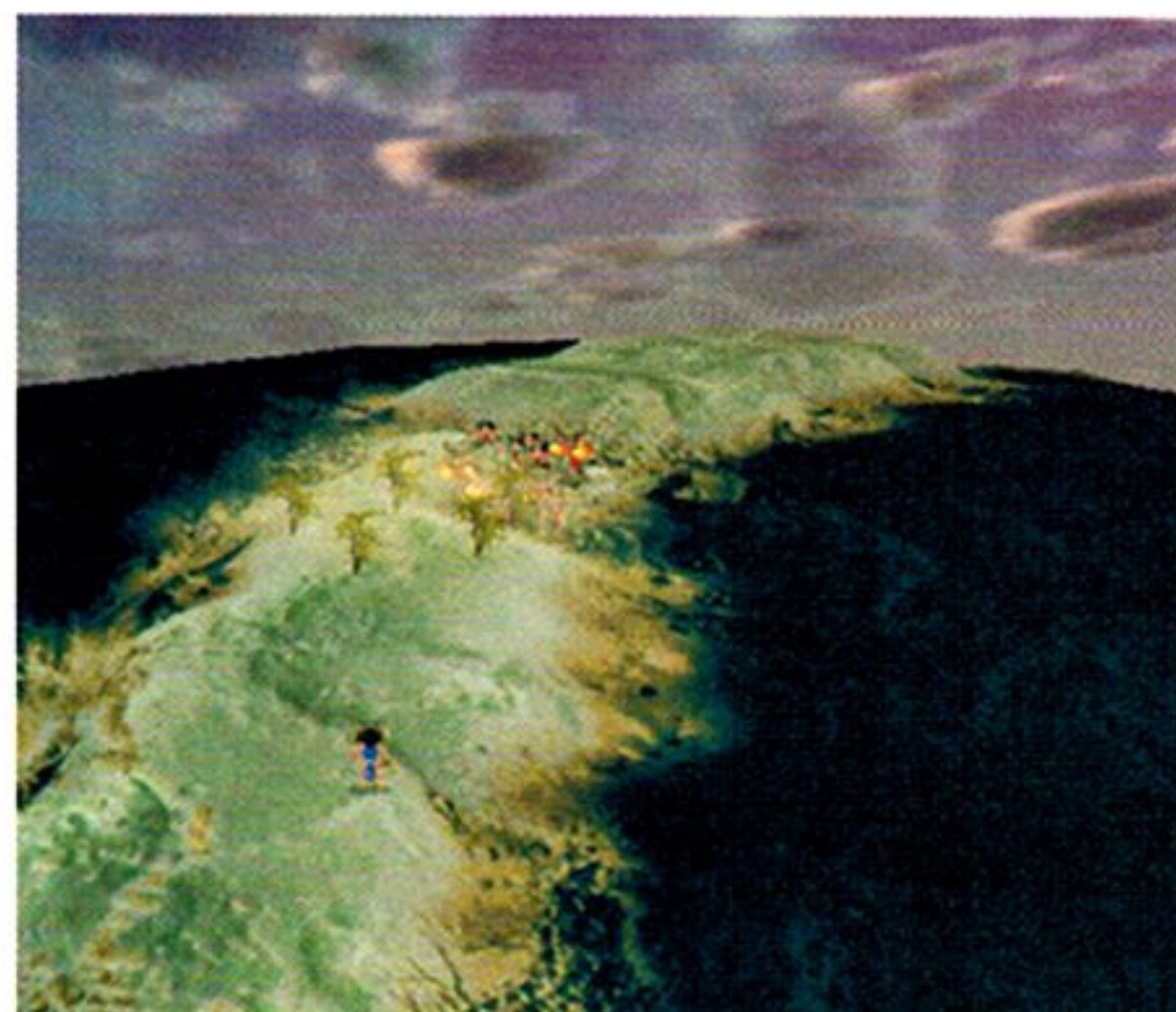
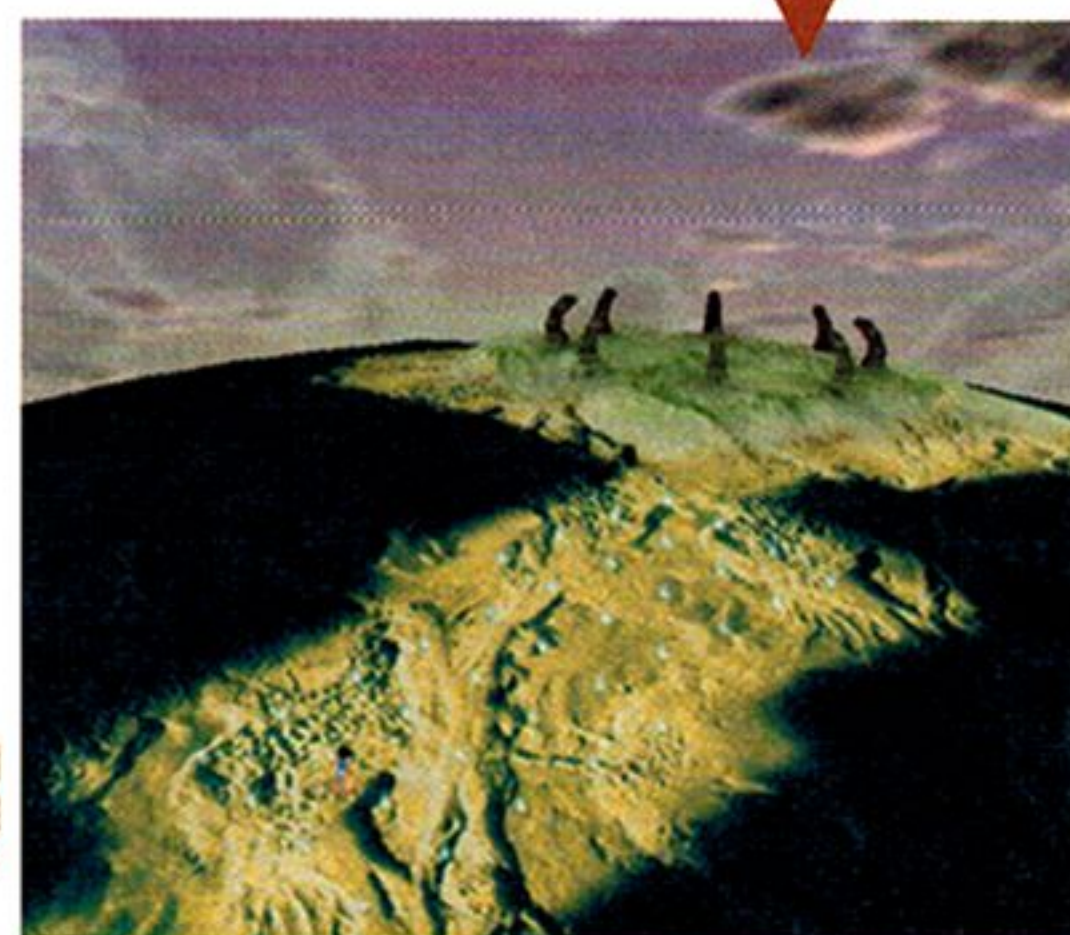
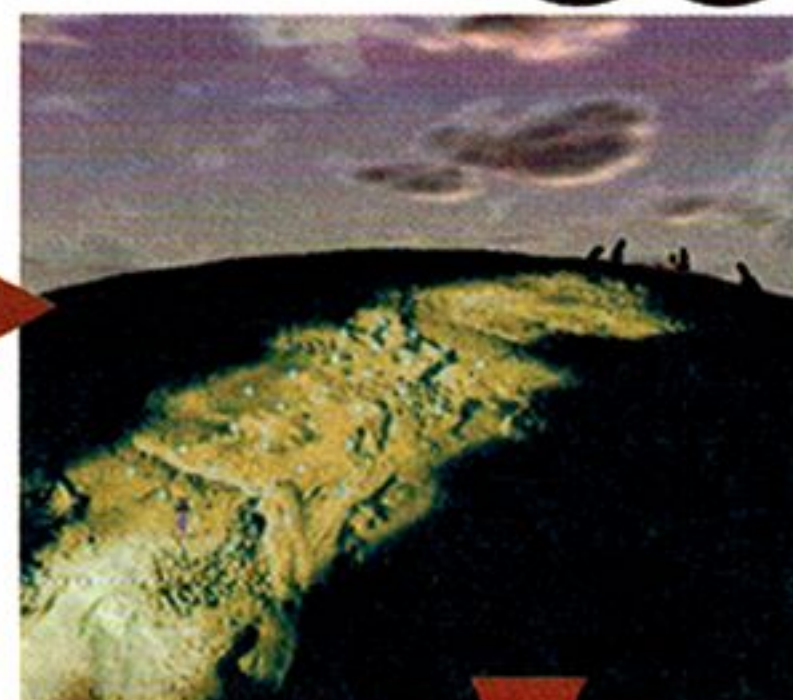
互いの「転生の地」の間には2つの中州があるので，結局3発の「土地隆起」スペ

島の東西の端はこのように防衛ラインを作り侵攻を簡単に許さないようにしておく

まずは島の中心部に移ってガードタワーを建設する

島と島は海が隔てているものの，「土地隆起」スペル一発で陸続きにしてしまえる

このレベルの概略図
1.「浸食」スペル(×3)

ルで繋げることができる。
ほとんどのプレイヤーがここからの侵攻を予測していないので、ここからの侵攻はかなり効果的な奇襲作戦になるはずだ。
なお、敵がここからの侵攻を企てているのに気づいたら、敵が造りだしたその陸路に対して「腐食」スペルを数発投じてやるといい。よほど人口が豊富でない限りここからの侵攻をあきらめてくれるはずだ。

このように中州を「土地隆起」スペルで繋いでいければ、

敵がこの戦法を取ってきたら……陸路に「腐食」スペルを敷き詰めてしまえばOK

自軍「転生の地」と敵「転生の地」を陸続きにすることができる

ふたつの島
TWO CRABS

心がまえ
互いに遠く離れた同条件の島が双方の陣地となる。平地は広いとはいえないが、建設スペースはそこそこにある。しかし、木々が少ないので、最低限必要な建物から建てていく必要がある。
それぞれの島にはオベリスクがあり、ここを礼拝すると「ボート小屋」の設計図が手に入る。島からでる手段はこのボートしかないので、とにかくこれを早く入手すること。
このレベルを使ったゲームは、速攻がきかない分、長期戦になりやすい。均衡状態に陥ると戦いが泥沼化して決着がつかなくなることもあるので「ハルマゲドン」などのゲストスペルを設定しておいたほうがいいだろう。

ボート小屋を早く建てる
双方の陣地は互いに離れており、攻撃を仕掛けるにはなんらかの乗り物を駆使する必要がある。
乗りものといえば「気球」「ボート」の2種類があるわけだが、初期値状態ではどちらも作ることができない。しかし、それぞれの陣地内には「ボート小屋」の設計図が隠されたオベリスクがあり、これを礼拝すると「ボート小屋」の設計図が入手できる。
まずはなによりも先にこれを礼拝すること。なお、オベリスクを礼拝できるのはシャーマンだけなので、その他の従者には建設の仕事をさせておこう。
ボート小屋を建てられるようになったら、陣地内の木々を使い果たす前に、ボート小屋を建てよう。そのほかの建物の建設を一時中断してもいいくらいだ。ボートが作れないと敵地へ攻撃することも、世界に点在する「ストーンヘッド」探索の旅にも出かけられないことになり、ゲーム展開がかなり不利になってしまう。
なお、ボート小屋は海岸に面した低地にしか建てられないので、建てられる場所は自ずと限られてくる。敵に侵攻されたときに、ボート小屋を破壊されないように防衛ラインを周囲に形成することも必要になってくる。

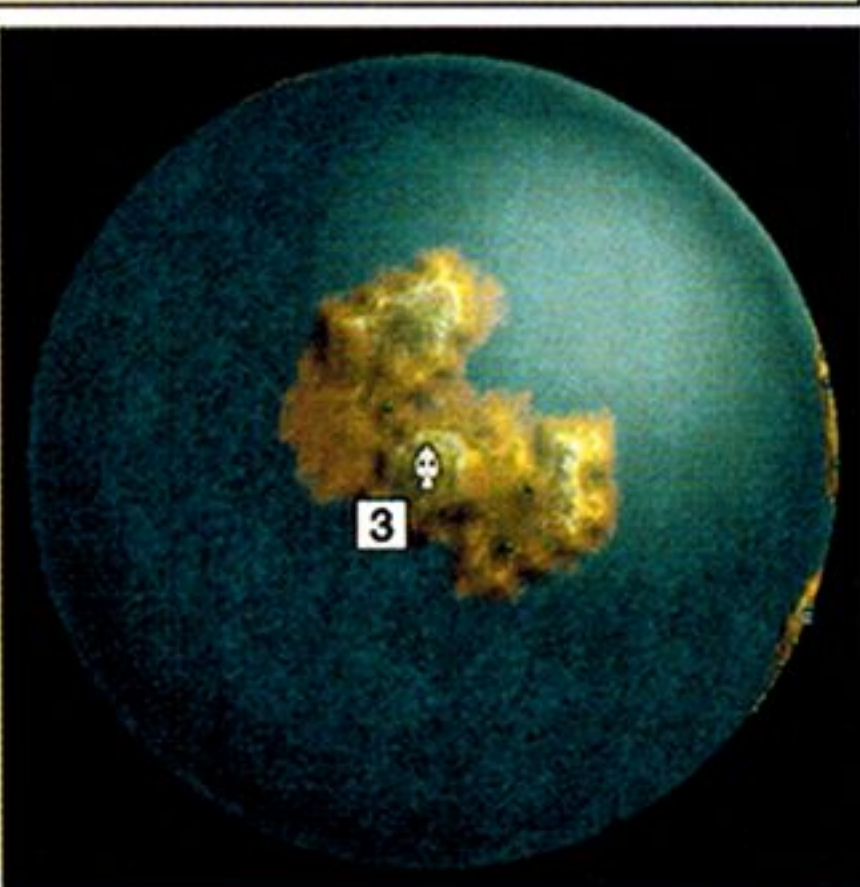
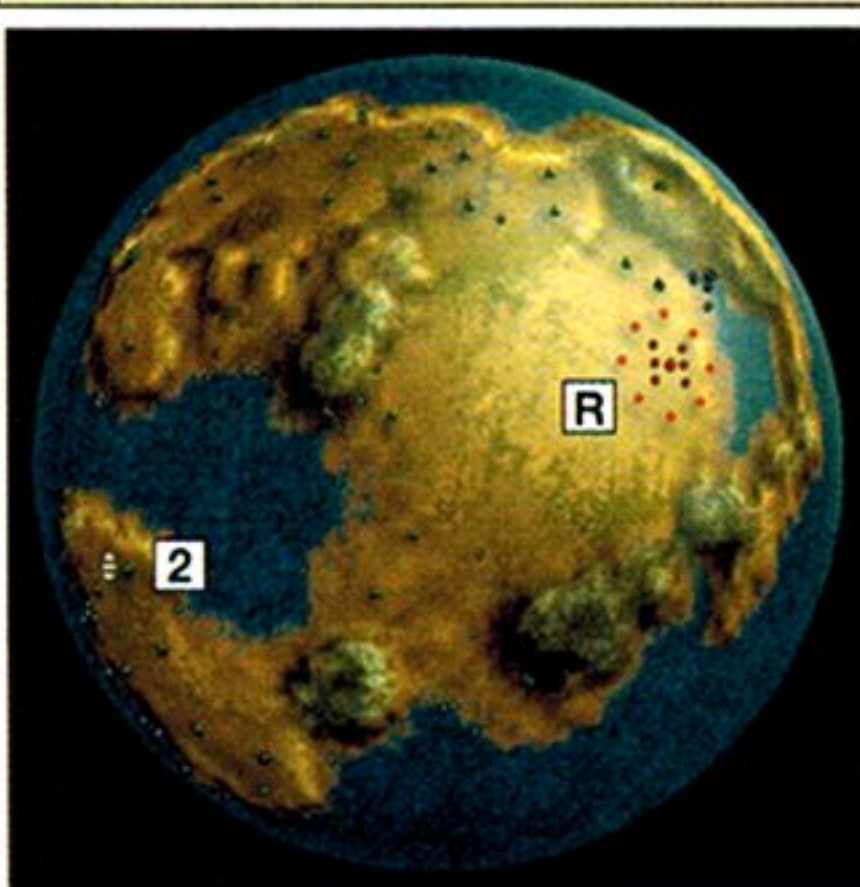
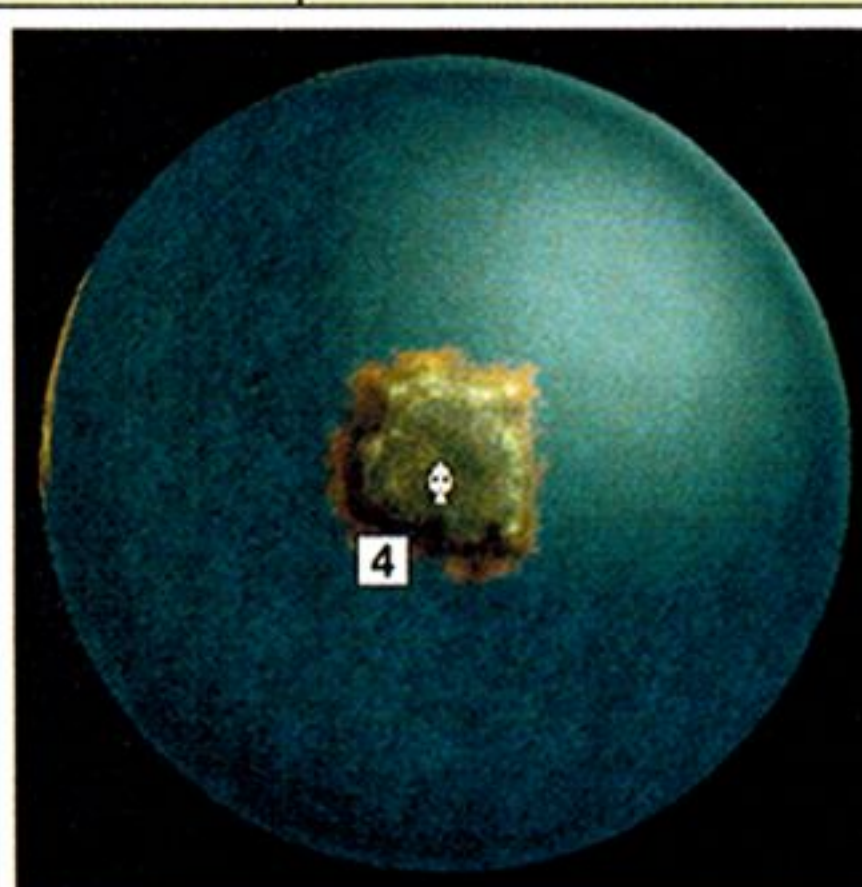
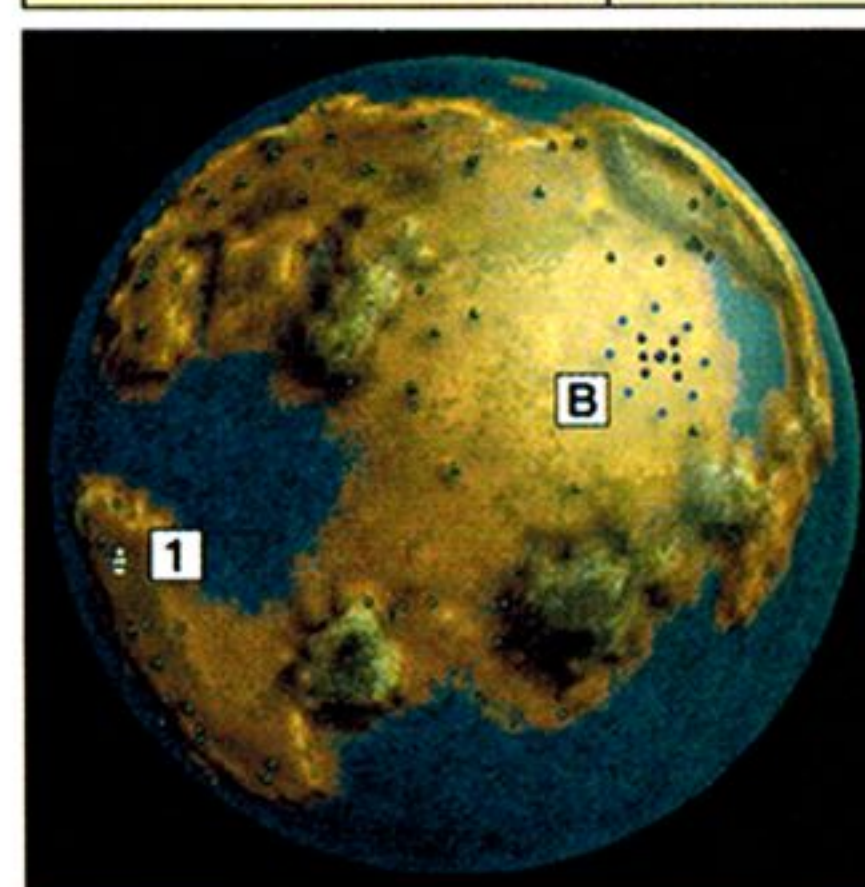
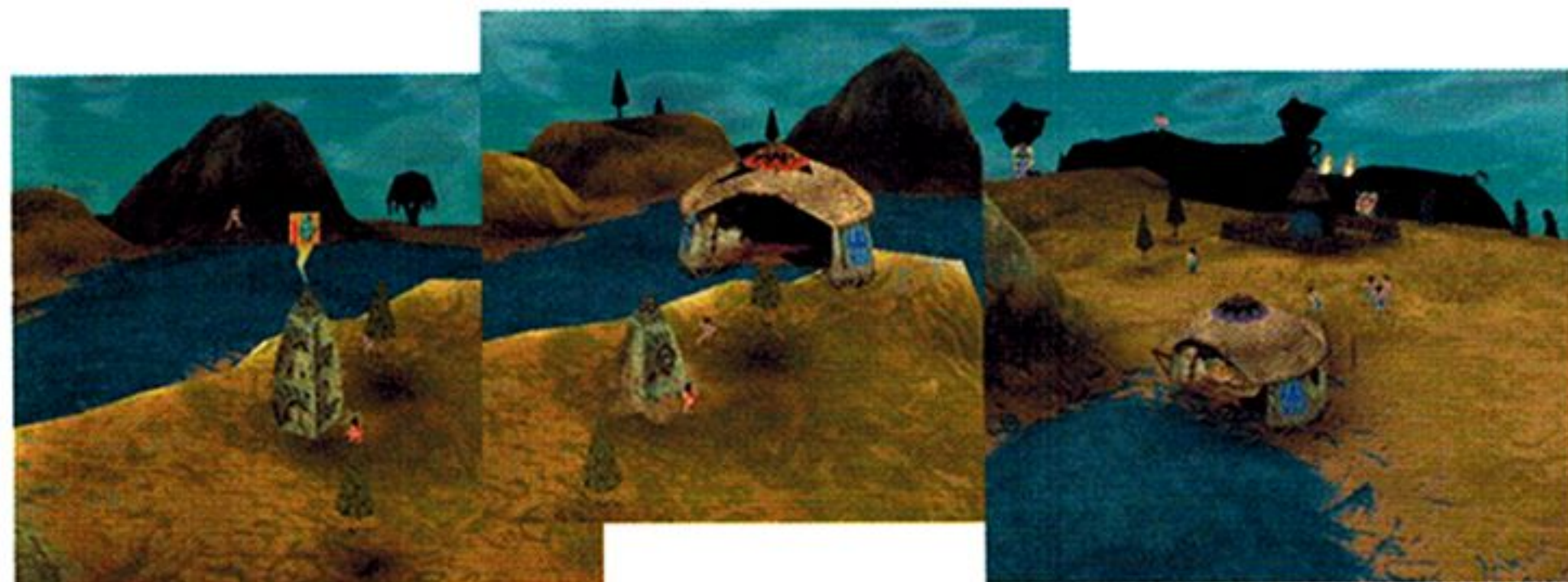
防衛ライン
このレベルの防衛方針とは、ズバリ「敵のボートを自軍の島に近づかせないこと」これに尽きる。
敵がボートで上陸できるポイントは何カ所かに限られるので、そのあたりを重点的に、島の外周にガードタワーを建てていくといいだろう。もちろんガードタワーには炎の戦士を住まわせておく。
ボートや気球に乗った従者は、炎の戦士のブラスト砲撃に非常に弱いため、海に向けて建てた炎の戦士入りのガードタワーは侵攻しようとする敵に対して強力なプレッシャーを与えることができるのだ。

最初から使えるスペル	
火山噴火	ライトニング
死の天使	土地隆起
ファイアストーム	マジックシールド
浸食	透明
地震	ハチの大群
土地平坦	幽霊軍隊
腐食	コンバート
トルネード	ブラスト
催眠	

最初から建てられる建物	
ガードタワー	炎の戦士訓練小屋
小屋	スパイ訓練小屋
伝道師訓練小屋	ガードポスト
戦士訓練小屋	

防衛ラインを崩すには
敵が島の外周に鉄壁の防衛ラインを形成してしまった場合、どうやって侵攻を仕掛けたらいいのだろうか。
いくつかの方法が考えられるが、いちばん単純なのが、侵攻させる従者に「透明」スペルや「マジックシールド」スペルをかけてしまう方法だ。「透明」スペルなら

このレベルの概略図			
1.「ボート小屋」の設計図	2.「ボート小屋」の設計図	3.「気球小屋」の設計図	4.マナのチャージを1500ポイント分、行ってくれる(回数は無制限。有効礼拝人数は8人まで)

B 1 4 R 2 3

海に面した島の海岸には均等にガードタワーを建て巡らせておくといい

敵地に近づいたら「死の天使」スペル発動

ば、まったく敵に気づかれずに、「マジックシールド」スペルならばガードタワーの炎の戦士のブラスト砲撃を跳ね返しながら敵地に上陸できる。

少しばかり危険と手間が伴うが、ボートに乗せたシャーマンを敵地に向かわせ、射程距離の長い「ライトニング」スペルで、ガードタワーを一軒一軒焼き払っていくという方法も考えられる。

多少コストが高くつくが、「死の天使」を使うのもいいだろう。

「死の天使」は自分に対して攻撃を仕掛けてくる炎の戦士を優先的に食い殺していく習性があるので、これを利用してやるのだ。

やり方だが、まず、敵の防衛ラインに適度に近づいてから「死の天使」スペルを発動してやる。すると一斉に敵防衛ラインのガードタワーの中の炎の戦士がこちらの「死の天使」に対してブラスト砲撃を開始するはずだ。「死の天使」はこれに怒り、彼らから優先的に食い殺してくれるのである。

「死の天使」はマナのチャージ量が多くコストが高くついてしまうが、その分プレイヤーは楽ができる。

防衛ラインが十分崩れたところを見計らって、一斉に軍勢を送り込めば、侵攻は成功だ。

「気球小屋」の設計図を与えてくれるストーンヘッド

ストーンヘッドは2つあり、いずれも自軍陣地からやや離れた小島にある。

比較的簡単に近づくことができるのが「気球小屋」の設計図を与えてくれるストーンヘッドだ。これはプレイヤー1（青）の陣地からは北方向、プレイヤー2（赤）の陣地からは東方向にある。

このストーンヘッドは、最初に礼拝した種族にのみ気球小屋の設計図を与え、そのあとは消えてしまうので、「早い者勝ち」ということになる。

つまり、早くボートを作って、これに乗ってここに移動できたものが先に礼拝できることになるのだ。繰り返しになるがそういった意味でもこのレベルではボート小屋を早く建設するということがとても大切なのだ。

さて、気球が作れるようになったからといって必ずしも「勝負に勝てる」というわけではないが、戦いを有利に運ぶことができるのは確かなので、ぜひとも手に入れたい。

もし、敵種族が先に祈っているのを発見したら、大至急駆けつけて、礼拝している敵従者に「催眠」スペルをかけ、そのまま礼拝を継続させて、「気球小屋」の設計図を横取りしてしまうといい。

島の東海岸側はでこぼこしているため侵攻ポイント向きではないが、敵地に面しているためその可能性は否定できない。ここにもガードタワーを何本か建てておこう

「死の天使」が防衛ラインを破壊している間に、プレイヤーは侵攻部隊を敵陣地の近くに集結させよう

マナをブーストしてくれる「ストーンヘッド」

このレベルには、礼拝することでマナのチャージを行ってくれるストーンヘッドが存在する。これも離れ小島に立っており、位置的にはプレイヤー1（青）の陣地から東方向、プレイヤー2（赤）の陣地からは北方向にある。

しかしストーンヘッドは高い絶壁の頂上に立っており、ただボートを寄せただけではストーンヘッドの前までたどり着けない。礼拝部隊にシャーマンを同行させ、絶壁の頂上に「土地隆起」スペルを投じて斜面を作り出して登るか、あるいは気球部隊を編成してここを訪れるか……いずれにせよ、なんらかの工夫をしなければ礼拝を始めることができないのだ。

仮に礼拝を始められたとしても、敵プレイヤーはこの礼拝を妨害してくるはずで、それなりの防衛策も立てておく必要があるだろう。

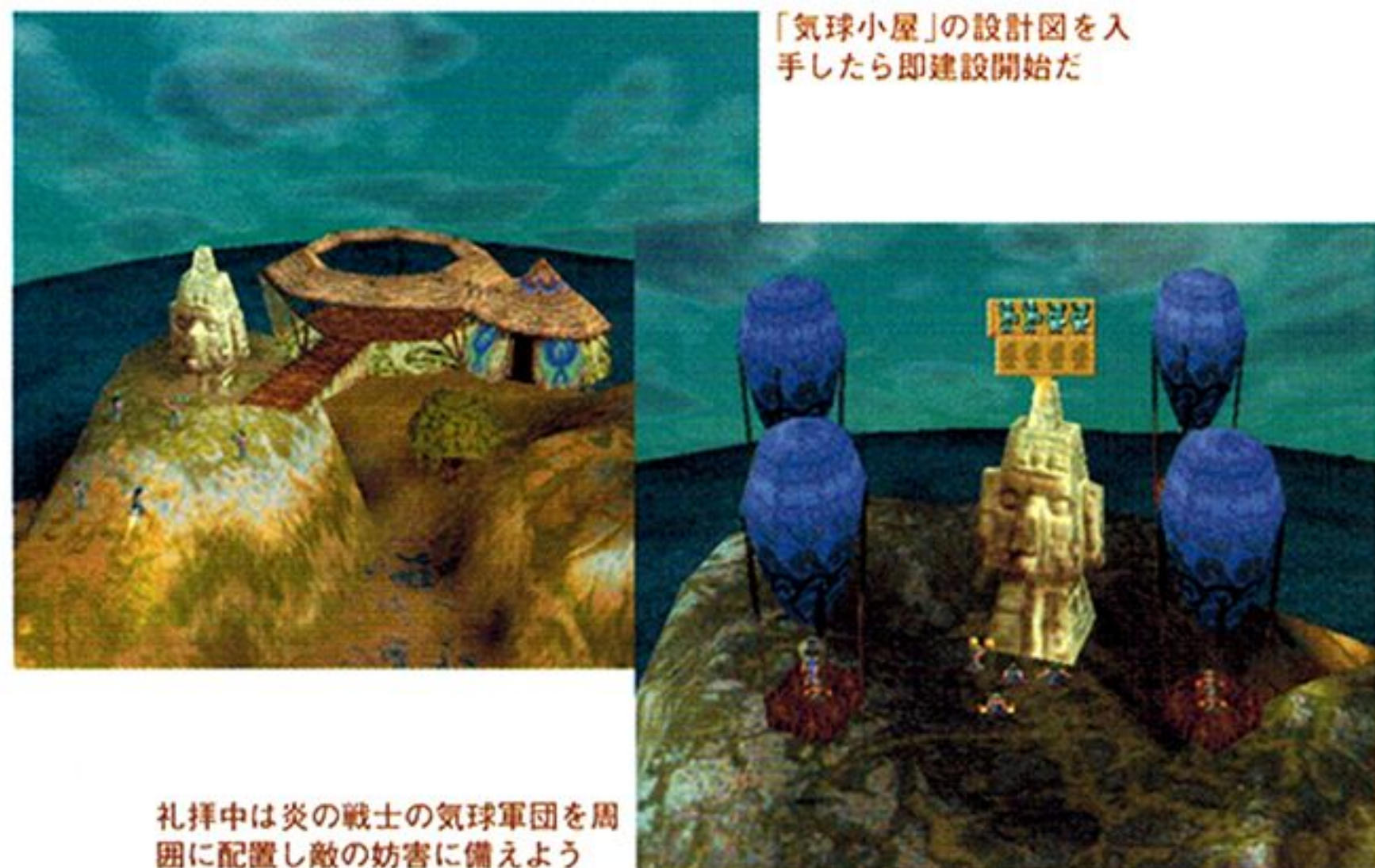
しかしこの絶壁には木がないのでガードタワーは建てられない。そこで炎の戦士を乗せた気球軍団をこのストーンヘッド周辺に配置するといい。

この絶壁はかなり高度があり、気球に乗っていることでさらに高度が高まる。よって炎の戦士のブラスト砲撃の射程距離が最大限に引き延ばされるのだ。

これにより敵の一般従者のボートによる接近をほとんど完璧に阻むことができ、この礼拝を妨害するためには敵プレイヤーはシャーマン自身をここに向かわせなければならなくなる。

「気球小屋」の設計図を与えてくれるストーンヘッドがある島。ボートがなければここにはたどり着けない

「土地隆起」スペルをストーンヘッドのもとへ投じれば、

絶壁が斜面に変わりストーンヘッドの前に行けるようになる

「気球小屋」の設計図を入手したら即建設開始だ

礼拝中は炎の戦士の気球軍団を周囲に配置し敵の妨害に備えよう

争い
SKIRMISH

最初から使えるスペル	
浸食	マジックシールド
腐食	透明
トルネード	ハチの大群
催眠	幽霊軍隊
ライトニング	コンバート
土地隆起	ブラスト

最初から建てられる建物	
ガードタワー	炎の戦士訓練小屋
小屋	気球小屋
伝道師訓練小屋	ガードポスト
戦士訓練小屋	

心がまえ
互いの陣地は狭く，しかも陸続きになっているため，短期決戦用として最適なレベルだ。

野人が少ないので，ゲーム序盤は「運命の丘」以上に激しい「野人争奪合戦」となるはずだ。

敷地面積が小さく木々もそれほど豊富ではないので建物は必要なものから建てていくこと。

互いの陣地中央にあるストーンヘッドは「地震」スペルを与えてくれるが，なかなか取らせてはもらえないだろう。もし敵が礼拝していたら，ためらわず妨害するか「催眠」スペルをかけて横取りしてしまうこと。

なにを建てるか
気球小屋が最初から建てられる点に注目する。

まずはこれを建て，同時進行で炎の戦士訓練小屋を建設する。両方が完成したら気球を作り，これに炎の戦士を乗せていく。

彼ら気球軍団を「移動可能な防衛ライン」として要所に配置するといい。

ガードタワーを建ててもいいのだが，木々があまり豊富ではないので，あまりおすすめできない。それよりはひとりで何役もこなせる気球軍団のほうがこのレベルでは使い勝手がいいはずだ。

短期決戦用レベルなので，敵がゲーム開始早々に伝道師部隊による速攻を仕掛けてくる場合もある。これに対抗するために，こちらも早いうちから伝道師小屋を作っておくことも忘れないように。

防衛ライン・その1
陣地の出入り口周辺は小高い丘になっているので，ここにガードタワーを建てて炎の戦士を住まわせる……というのが「まとも」な戦略といえるだろう。ただし，気球部隊を編成しているならば，この小高い丘に気球を停泊させておくだけで防衛ラインになりうる。

防衛ライン・その2
互いの島が非常に細い道で陸続きになっている点に注目する。敵は陸路から侵攻する場合，この細い道を通るしかないのだ。

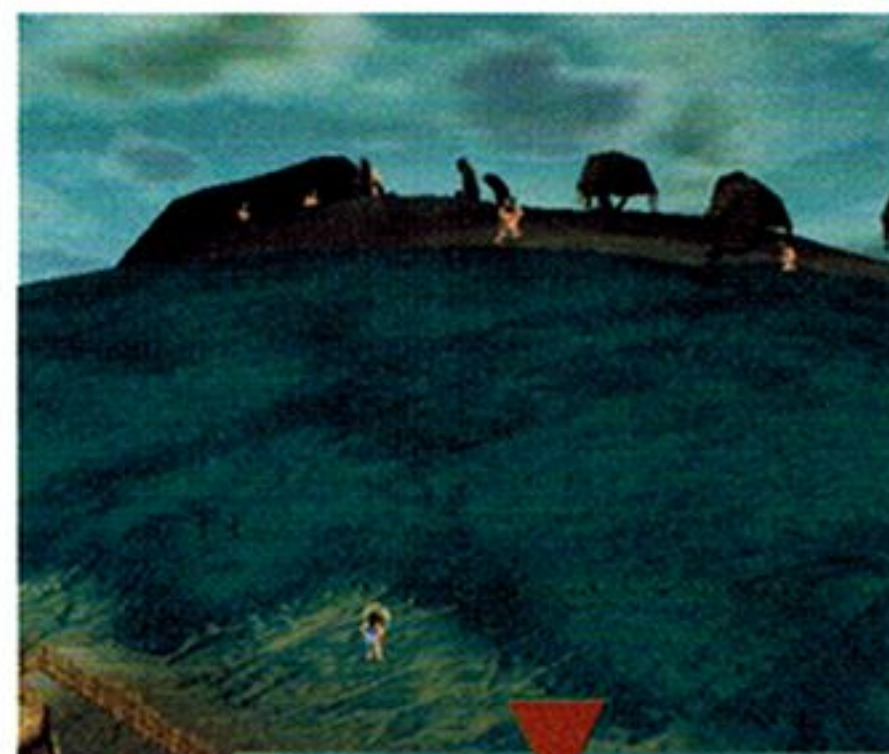
そこで，早くから「腐食」スペルをチャージしておき，これをストーンヘッドのある丘を越えた敵地側の道へ何発か仕掛けてしまうのだ。これで敵はこちらへの侵攻手段を失うことになる。

なお，こちらからこの陸路を使って侵攻を仕掛けるときは，「腐食」スペルをかき消すことを忘れないように。やり方は仕掛けた「腐食」スペルの上で[Shift]＋右クリックだ。

ストーンヘッド
ストーンヘッドは「地震」スペルを与えてくれる。敷地の狭いこのレベルにおいては「地震」スペルは勝負決定づける強力なスペルといえる。

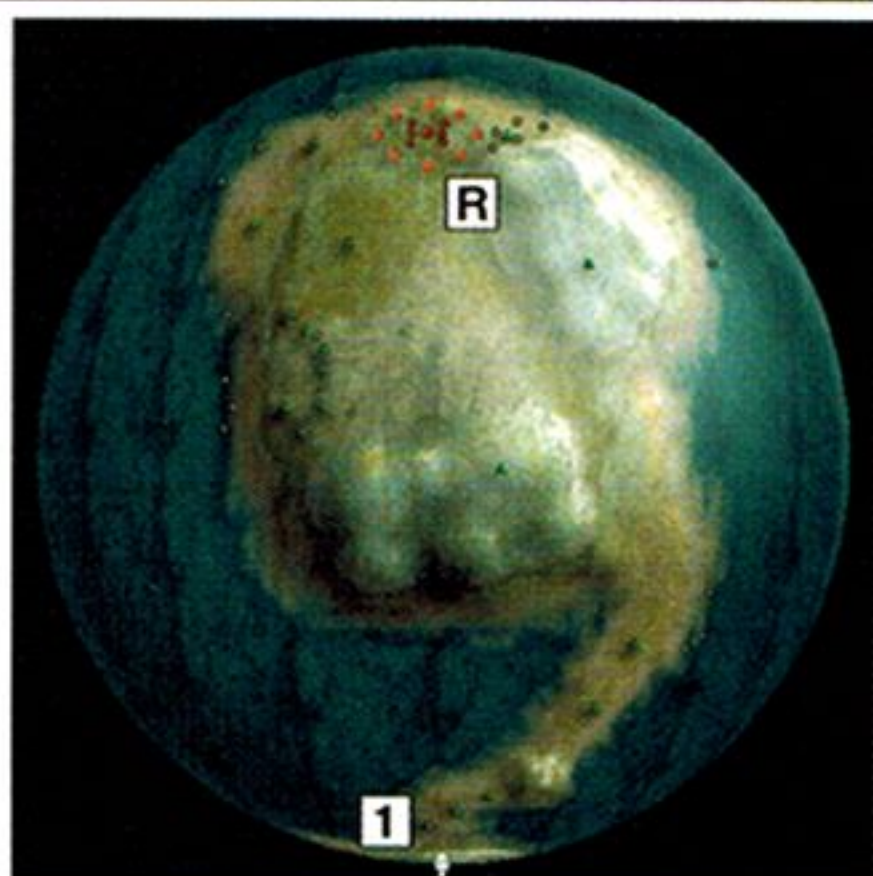
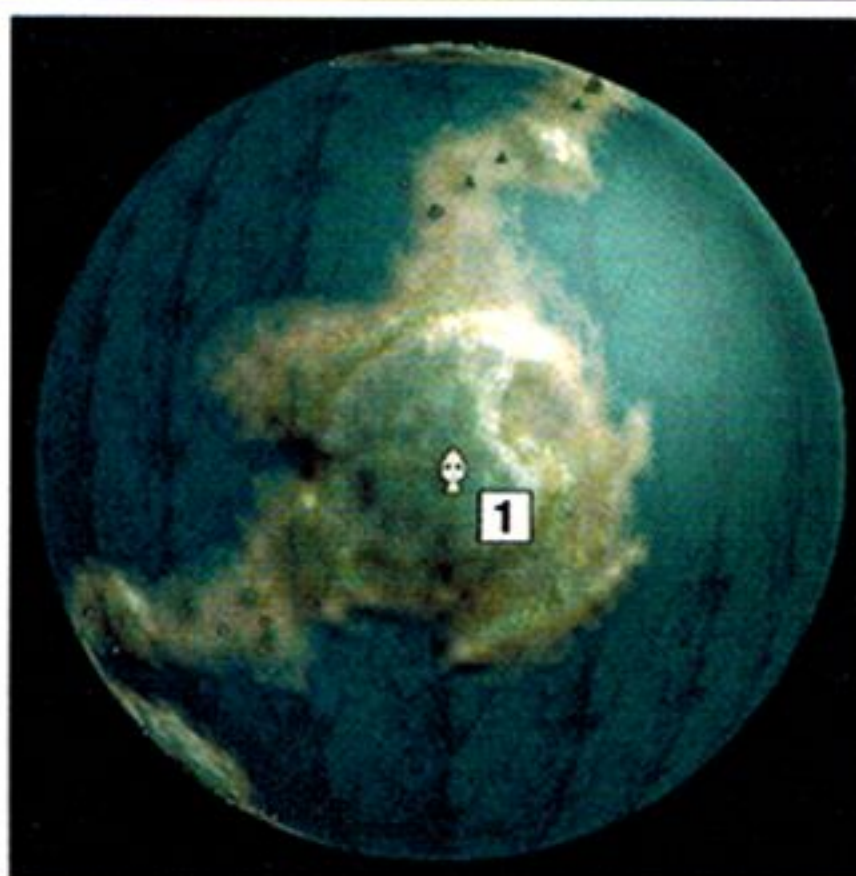
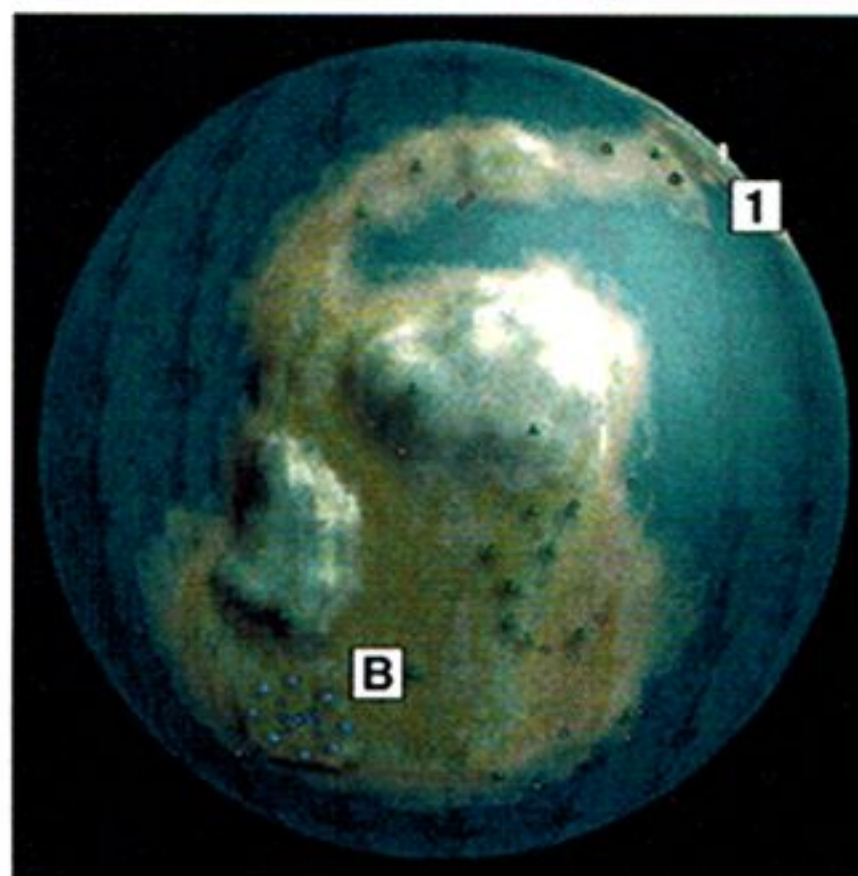
ぜひとも自軍が取りたいし，敵軍には取らせたくないものだ。

この礼拝を成功させるには，「防衛ライン・その2」で紹介した，ストーンヘッドのある丘と敵陣地の間の細い道に「腐食」スペルをしかける戦略が有効だ。これで敵はストーンヘッドに近づけなくなる。

さらに，このストーンヘッドが小高い丘にある点に注目し，気球軍団をここに集結させるという作戦もいい。小高い丘の上に浮かんだ炎の戦士の気球軍団はかなり遠くの敵に対しても先制攻撃が仕掛けられるので，敵はストーンヘッドにまったく近づけないことになる。

気球軍団はストーンヘッドのある丘の頂上の敵地側方面に集結させておくといい

防衛ライン　その1

防衛ライン　その2

このレベルの概略図
1.「地震」スペル（×3）

「裏」侵攻ポイント

もし、敵が、ここであげた作戦を実施してきた場合、こちらはまったく侵攻できなくなってしまう。では、どうしたらよいのか。

まあ、気球が使えるので空路を使って攻撃を仕掛けるというのがもっとも単純な対応策といえるだろう。

しかし、もうひとつ裏をかいた作戦がある。

「ワールドビュー」を見てもらうとわかるが、プレイヤー1(青)の陣地からは南東に、プレイヤー2(赤)の陣地からは北西に、互いの敵陣が海の向こうに見えるはずだ。なんと大胆にも、海の向こうの敵陣に向け「土地隆起」スペルを仕掛けていき、敵陣とを陸続きにしてしまうのだ。

敵は海の向こうからの侵攻に対する防衛ラインはほとんど形成できていないはずなので、かなりの高確率でこの奇襲作戦は成功するはずだ。

なお、敵地と陸続きになるということは、敵もそこから進撃してくるかもしれないということを忘れないように。

なお、陸続きにするまでには「土地隆起」スペルは4発は必要だ。「土地隆起」スペルが1発ストックされるごとに「土地隆起」スペルを使っていったのでは、作戦がバレバレなので、必ず4発ストックしてから連続発動させること。

果てしない世界
ALL AROUND THE WORLD

心がまえ

このレベルも互いの陣地が陸続きになっている。しかし、非常に全長の長い島の両端に互いの陣地があるために、両軍の接触にはかなりの時間がかかる。そういう意味では、陸続きでありながらも長期戦になりやすいレベルといえるだろう。

比較的野人は数が多く、木々もそこそこにあるため、建築と人口増加はかなりスムーズに進むはずだ。

なお、両軍の地形は対称でないので、戦略によっては若干の有利不利が発生する可能性がある。

野人争奪戦

互いの陣地は遠く離れているため、自軍の陣地内の野人が奪われる可能性は低い。しかし、それぞれの陣地を結んでいる長い長い陸地の上にいる野人は、両軍ともに獲得できるチャンスがあるため、ゲーム序盤はこれの奪いあいになることだろう。

野人は水辺や海岸線に集まりやすいので、そういったところを探し回るのは基本中の基本だ。

なお、このレベルに限っては写真に示した、谷間の池が野人であふれ返っている。ゲーム開始直後はまずここに急行するといい。

この池はプレイヤー2(赤)の陣地からのほうが若干近いようだ。プレイヤー2(赤)の立場からすれば、プレイヤー1(青)に取られる前に、この池の野人をまず全部コンバートしてしまったほうがいいだろう。

防衛ライン

それぞれの陣地の周囲には小高い丘が点在している。そういった丘の頂上にガードタワーを建てて炎の戦士を住まわせておくといいだろう。

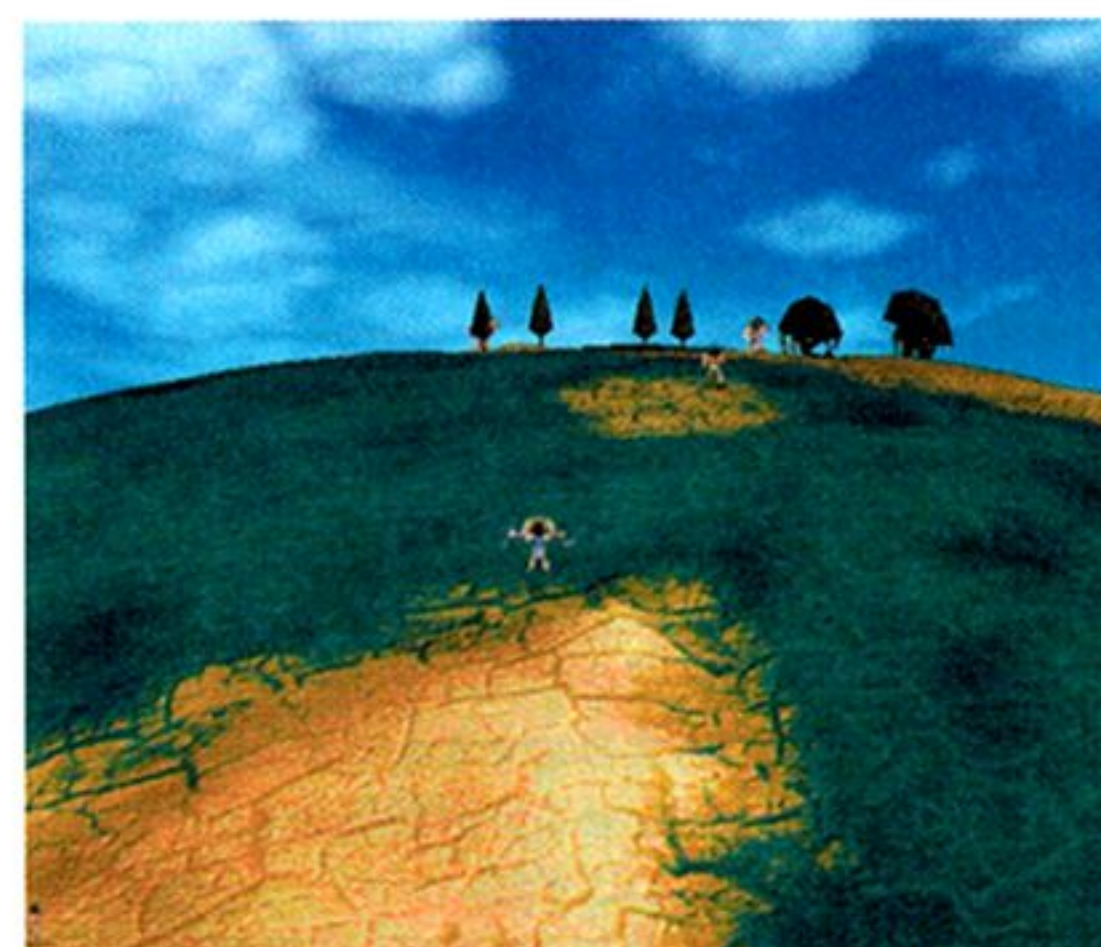
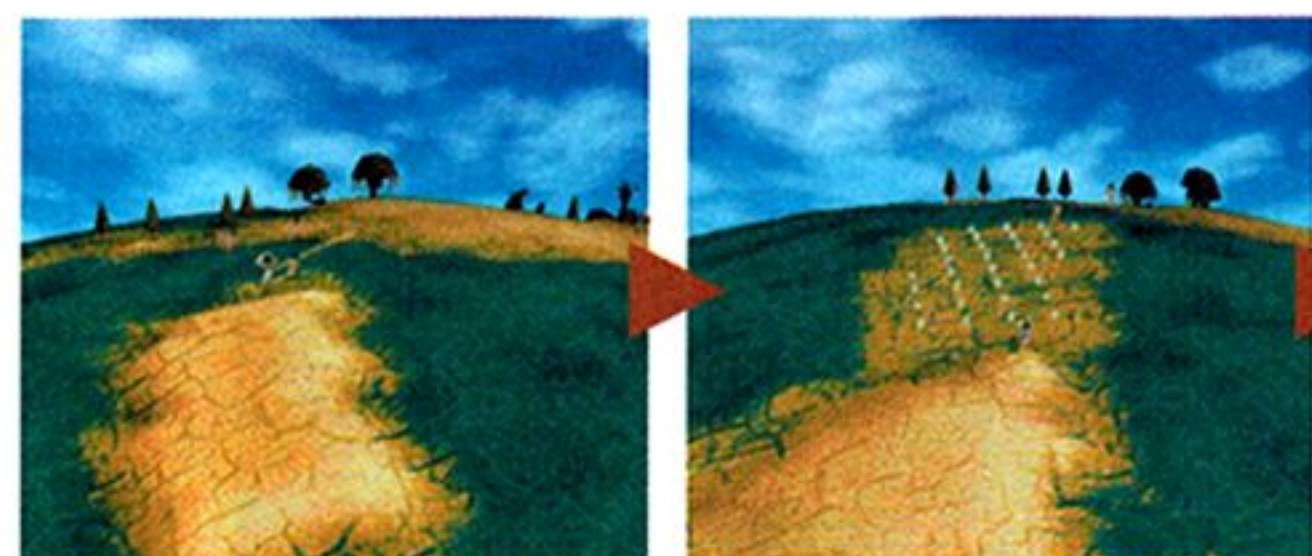
防御上有利なのは、プレイヤー2(赤)の陣地で、敵地に面した北側には幅の広い高い丘が東西に伸びていて、さらにそのふもとには堀状の池まである。プレイヤー2(赤)はまずはここに防衛ラインを形成することから始めるべきだろう。

一方、プレイヤー1(青)は前段で紹介した野人の群がいる谷間の池の南側の丘にガードタワーを建て巡らせることを考えよう。ここはかなりプレイヤー1(青)本拠地から離れているが、これより北側(プレイヤー1

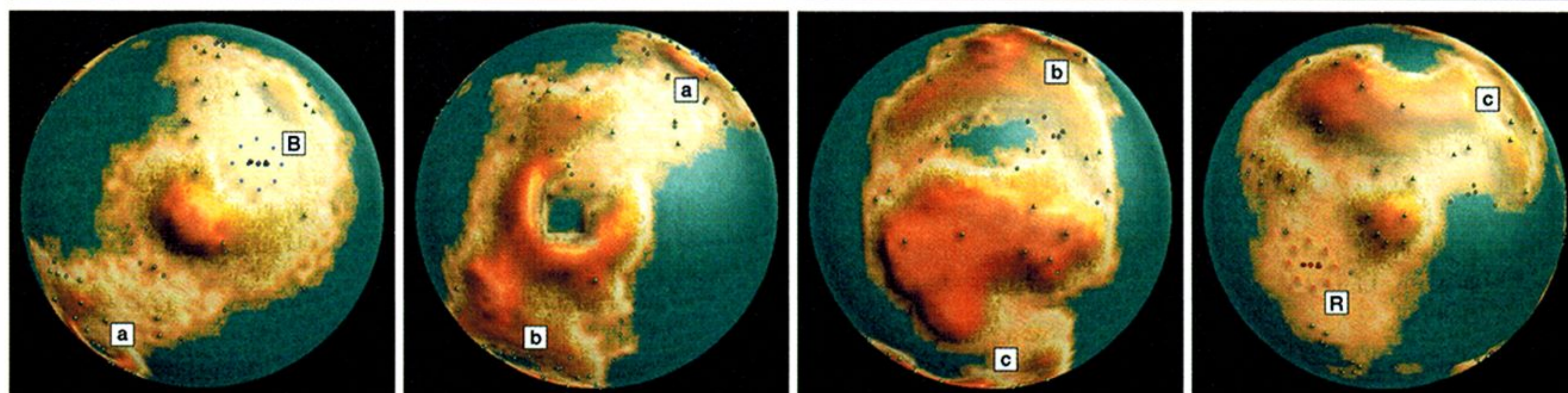
最初から使えるスペル	
浸食	マジックシールド
腐食	透明
トルネード	ハチの大群
催眠	幽霊軍隊
ライトニング	コンバート
土地隆起	ブラスト

最初から建てられる建物	
ガードタワー	炎の戦士訓練小屋
小屋	スパイ訓練小屋
伝道師訓練小屋	ボート小屋
戦士訓練小屋	ガードポスト

このように両軍のシャーマンがここで出くわすことも多いはず

「裏口」侵攻ポイントにはこのような中州がある

2発の「土地隆起」スペルがあれば敵地までのショートカットが作れてしまう

「ワールドビュー」で見るとわかるが、互いの陣地の本拠地に通ずる「裏口」が用意されていることがわかる

このレベルの概略図

(青)陣地に近い側)には防衛ラインを形成するのに適したポイントがないのだ。
　ゲーム開始直後に獲得したばかりの勇士をいきなりこの地へ送り込み、ガードタワーの建設を始めさせてもいいくらいだ。

プレイヤー1(青)はぜひともここを占拠したい

プレイヤー2(赤)の陣地近くには防衛に適した丘がある

カウンターアタック

「陸続き」という印象が強いこのマップにも実は裏口が用意されている。
「裏口」は2つあり，プレイヤー1(青)の陣地を基準にして位置を示すとすると，プレイヤー1(青)の「転生の地」より北西側と南西側にある。
　さて，それではいったいどういう裏口なのか。
　写真を見てほしい。意味ありげな中州に気が付いただろうか。
　この中州に「土地隆起」スペルを投じ，さらにこの中州に渡ってから向こう岸にもう1発「土地隆起」スペルを投ずれば敵陣本拠地のすぐ近く(あるいは敵陣のど真ん中)に侵攻路を作り出すことができるのだ。
　本来ならば長い道のりを歩いて侵攻しなければならないところを，このショートカットを作り出してしまえば，ごく短い距離で，敵地に行けてしまうのだ。
　しかもこの橋渡しに必要なのは「土地隆起」スペルたった2発だけ。スペルを投じているのを見てから対応するのはまず無理なので，奇襲作戦としてはかなり効果が高いはずだ。
　守る側としては見てから対応するのはまず無理なので，あらかじめこの奇襲作戦を予測して，防衛ラインを形成しておく必要があるだろう。

カウンターアタックに備え，中州に面した海岸には防衛ラインを形成しておくといい

3プレイヤーマップの完全攻略

島々
LINKED ISLES

最初から使えるスペル	
浸食	ライトニング
地震	土地隆起
土地平坦	ハチの大群
腐食	幽霊軍隊
トルネード	コンバート
催眠	ブラスト

最初から建てられる建物	
ガードタワー	炎の戦士訓練小屋
小屋	スパイ訓練小屋
伝道師訓練小屋	ガードポスト
戦士訓練小屋	

心がまえ

　3種族それぞれが異なった地形条件の陣地を持ってスタートする。
　プレイヤー1(青)は比較的平地が広いので建物が建てやすく，プレイヤー2(赤)は高地が多いので防衛状条件に恵まれている。なお，プレイヤー1(青)とプレイヤー2(赤)の陣地は同じ大陸の南北の両端に位置しており，互いにかなり接触しやすい位置関係にある。
　プレイヤー3(黄)はひとつの大陸を独り占めしている状態で，この大陸はあとの2人のプレイヤーがいる大陸と細い陸路で接続されている。
　プレイヤー1(青)とプレイヤー2(赤)の激しい戦いを傍観しつつ自軍の勢力を拡大するのがプレイヤー3(黄)……といった展開になりやすい。
　たとえば，防衛上，地形的条件に恵まれているプレイヤー2(赤)は自分たちにはほとんど役に立たない「ボート小屋」が得られ，攻め立てられやすいプレイヤー1(青)は進撃時に非常に役に立つ「マジックシールド」が得られるのだ。プレイヤー3(黄)は進撃の際に非常に役に立つ「透明」スペルが得られるわけだが，プレイヤー1(青)やプレイヤー2(赤)の陣地がある大陸へ侵攻する際には細い陸路を渡らなければならず，ここに敵によっ

このレベルの概略図
1.「透明」スペル(回数無制限)
2.「マジックシールド」スペル(回数無制限)
3.「ボート小屋」の設計図(各種族1回ずつ)
4.～6. マナのチャージを500ポイント分，行ってくれる(回数無制限。有効礼拝人数は6人まで)

オベリスクをめぐる争い

　それぞれの陣地の地形的条件が異なっているのはすでに述べたとおりだが，それだけでなく，陣地内にあるオベリスクが与えてくれるものも異なるのだ。
　どの種族の陣地になにを与えてくれるオベリスクがあるのかを，まず覚えておこう。
プレイヤー1(青)…
　「マジックシールド」スペルの知識
プレイヤー2(赤)…
　「ボート小屋」の設計図
プレイヤー3(黄)…
　「透明」スペルの知識
　こうして見比べると，絶妙なバランス設定がなされていることに気づく。

て「腐食」スペルを仕掛けられたらそこまでだ。この陸路を使わないで進撃するにはボートを作るしかない。

このように、どうも、お互いの地形的条件の有利不利を調整する格好でオベリスクが設置されているようなのだ。

自軍の戦闘能力を完璧にするためには敵地のオベリスクを礼拝する必要があるわけで、ここにこのレベルの「戦いの根元」のようなものが作り出されているといえる。

なお、それぞれのオベリスクは礼拝するとスペルの知識や建物の設計図を与えてくれるので一度礼拝してしまえば、同じオベリスクを礼拝する必要はない。

ストーンヘッドをめぐる争い

各陣地にはマナをチャージしてくれるストーンヘッドが1個ずつある。

ストーンヘッドを炎の戦士を住まわせたガードタワーで囲んでしまうといい。このストーンヘッドは「そこまでして守る」価値はある

このストーンヘッドは礼拝が完了するたびにマナを500ポイント分チャージしてくれ、しかも回数は無制限だ。強力な攻撃スペルのチャージを敵よりも早く完了するためにもこれはゲーム開始直後から礼拝すべきだろう。

逆にいえば、この礼拝を妨害できれば敵を出し抜くことができるということでもある。

そこで、自軍のこのストーンヘッドの礼拝を邪魔されないようにする工夫と、敵陣地のこのストーンヘッドの礼拝を邪魔する作戦の2つを考えることになるわけだ。

まず、邪魔されないようにする工夫だが、このストーンヘッドは高い丘の上にあるという点に着眼し、この丘の上にガードタワーを建て炎の戦士を住まわせ、防衛ラインを形成してしまうといい。このガードタワーの中の炎の戦士はこのストーンヘッドに近づこうとする敵を撃退してくれるはずだ。

一方、邪魔をする作戦だが、これは伝道師を送り込むのがいいだろう。多くのプレイヤーがこのストーンヘッドに礼拝させているしているのは勇士のはずだ。これをこちらの伝道師で寝返らせてしまおうというわけだ。もし、敵が、この礼拝を邪魔されないようにという目的で、前述したような防衛ラインを築いていた場合には、送り込む伝道師には「マジックシールド」スペルや「透明」スペルをかけて、防衛ラインの目を盗んで接近させる必要があるだろう。

ところで、このストーンヘッドにはどんな従者を礼拝させるのがいいのだろうか。

「礼拝を邪魔する作戦」として「伝道師を送る」というのを紹介しているが、この可能性を考慮すれば、伝道師に対抗できるのは伝道師だけなので、伝道師を礼拝させるのがいちばんだということになるだろう。

防衛ライン

防衛ラインは、各プレイヤーの地形条件が異なるため、それぞれが異なったものを形成する必要がある。

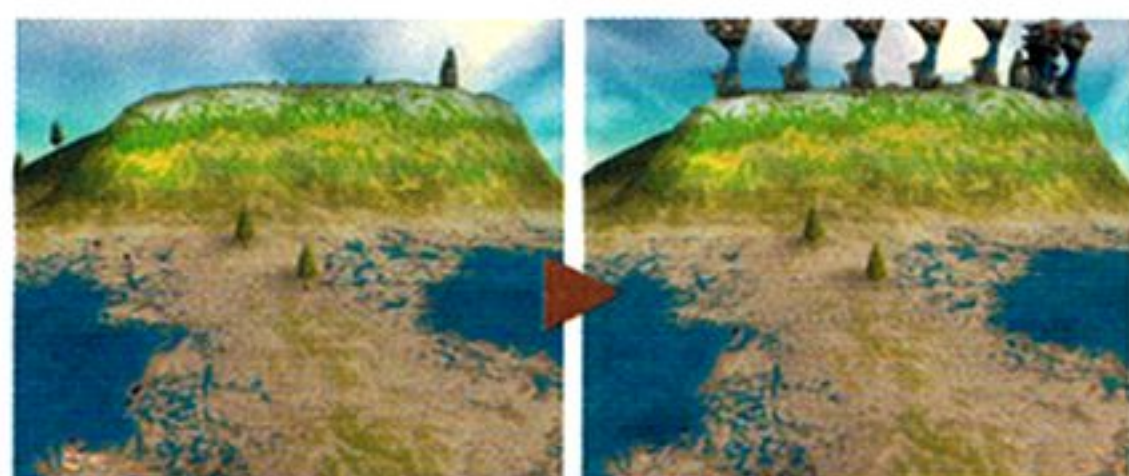
まず、プレイヤー1(青)だが、オベリスクが立っている小高い丘と、その西側の小さめの丘にガードタワーを建てて炎の戦士を住まわせるといい。これは主にプレイヤー2(赤)の侵攻対策ということになるが、プレイヤー3(黄)の侵攻の可能性も考慮して、オベリスクが建っているほうの丘の東側、プレイヤー3(黄)の陣地に繋がる陸路に面した側にも建てたほうがいいだろう。

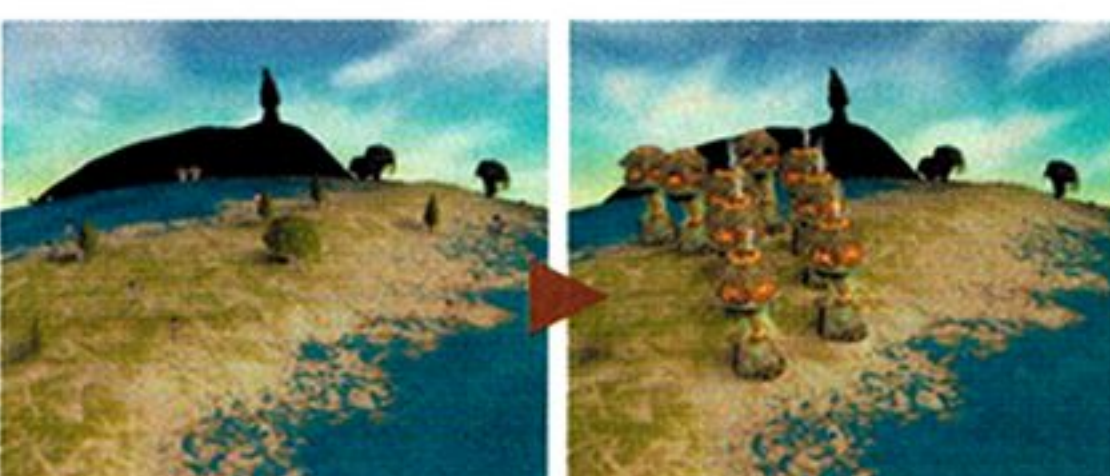
プレイヤー2(赤)の場合は、まず、「転生の地」東側の、プレイヤー3(黄)の陣地に接続している陸路を遮断する格好で形成することから始めよう。続いて南西側一帯に広がる高地の、プレイヤー1(青)の陣地に面した端に建て並べる。これはいうまでもなくプレイヤー1(青)の侵攻対策だ。

プレイヤー3(黄)は、プレイヤー1(青)とプレイヤー2(赤)の陣地がある大陸に向けて伸びている2つの陸路をふさぐ格好でガードタワーを建て、ここに炎の戦士や伝道師を配置するといい。もし「腐食」スペルのストックがあるならば、この防衛ラインの向こう側の陸路の上に仕掛けてしまうといい。防衛ラインと「腐食」スペルの2段構成の防衛ラインはそう簡単に破れるものではない。

プレイヤー1(青)の防衛ライン・その1。こちらは対プレイヤー2(赤)防衛ラインとなる

プレイヤー2(赤)の防衛ライン・その1。こちらは対プレイヤー1(青)防衛ラインとなる

プレイヤー3(黄)の防衛ライン・その1。こちらは対プレイヤー1(青)防衛ラインとなる

プレイヤー1(青)の防衛ライン・その2。こちらは対プレイヤー3(黄)防衛ラインとなる

プレイヤー2(赤)の防衛ライン・その2。こちらは対プレイヤー3(黄)防衛ラインとなる

プレイヤー3(黄)の防衛ライン・その2。こちらは対プレイヤー2(赤)防衛ラインとなる

争い
SKIRMISH

心がまえ

2プレイヤーの同名のレベルを3プレイヤーに拡張したもの……というイメージだ。基本戦略は2プレイヤーのものがそのまま使えると考えていい。

地形的には全プレイヤーがほぼ同条件と考えていいが，プレイヤー3(黄)のみ、若干防衛上いい位置にいる(後述)。

基本戦略

基本戦略は2プレイヤーの同名レベルのものがそのまま使える。簡単にまとめると、

- **敷地が狭く木々も少ないので必要な建物から建てていくこと**
- **野人も少ないのでゲーム序盤は貪欲に野人獲得に奔走すること**
- **「地震」スペルを与えてくれる中心部のストーンヘッドはぜひとも奪取すること。先に敵に礼拝された場合は必ず妨害すること**
- **互いの陣地を結んでいる陸路に「腐食」スペルを投じておけば，敵の侵攻を防ぐことができる**

といった感じになる。

しかし、気球小屋が作れないために，2プレイヤー版「争い」で使えた気球軍団による「移動可能な防衛ライン」作戦が使えない。よってこのレベルにおける防衛ラインはガードタワーで作って形成するしかないのだが，すでにいったように木々が少ないので建てる軒数とタイミングが難しい。

最初から使えるスペル	
浸食	マジックシールド
腐食	透明
トルネード	ハチの大群
催眠	幽霊軍隊
ライトニング	コンバート
土地隆起	ブラスト

最初から建てられる建物	
ガードタワー	炎の戦士訓練小屋
小屋	気球小屋
伝道師訓練小屋	ガードポスト
戦士訓練小屋	

ストーンヘッド

このレベルのストーンヘッドは2プレイヤー版同様に「地震」スペルを3発与えてくれる。2プレイヤー版でも述べたがこの狭い敷地スペースにおける「地震」スペルは

決定打となり得るので，絶対敵に取らせてはならない。敵に取られるくらいならば，ストーンヘッドに「腐食」スペルを仕掛けてしまい，誰にも取れないようにしてもいいくらいだ。

プレイヤー3(黄)は「裏口」が使えない

２プレイヤー版では礼拝中に気球軍団を停泊させて護衛にあたらせる作戦を紹介したが，3プレイヤー版では思い切ってここにガードタワーを建ててしまうのもいいかもしれない。

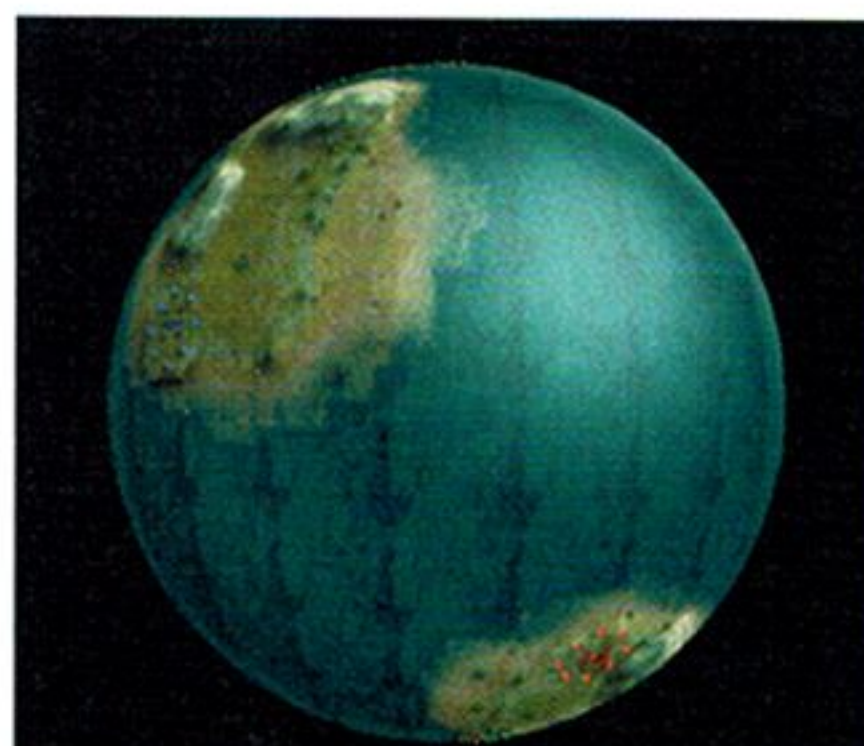
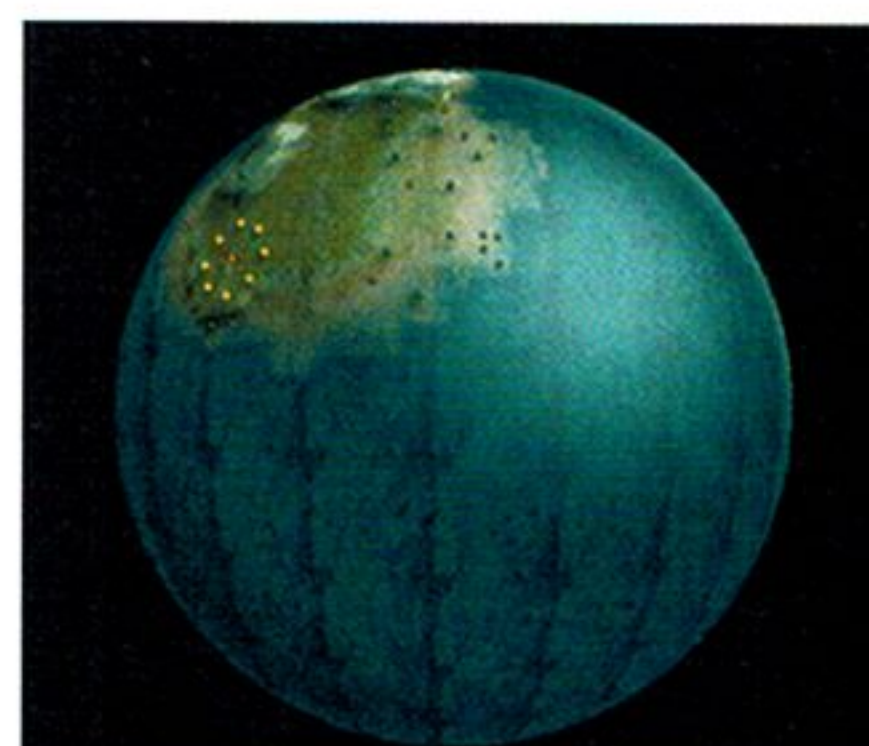
ここに炎の戦士，あるいは伝道師を住まわせておけば，礼拝中は，礼拝者の護衛として活躍し，敵が侵攻してきたときは防衛ラインとして機能するからだ。

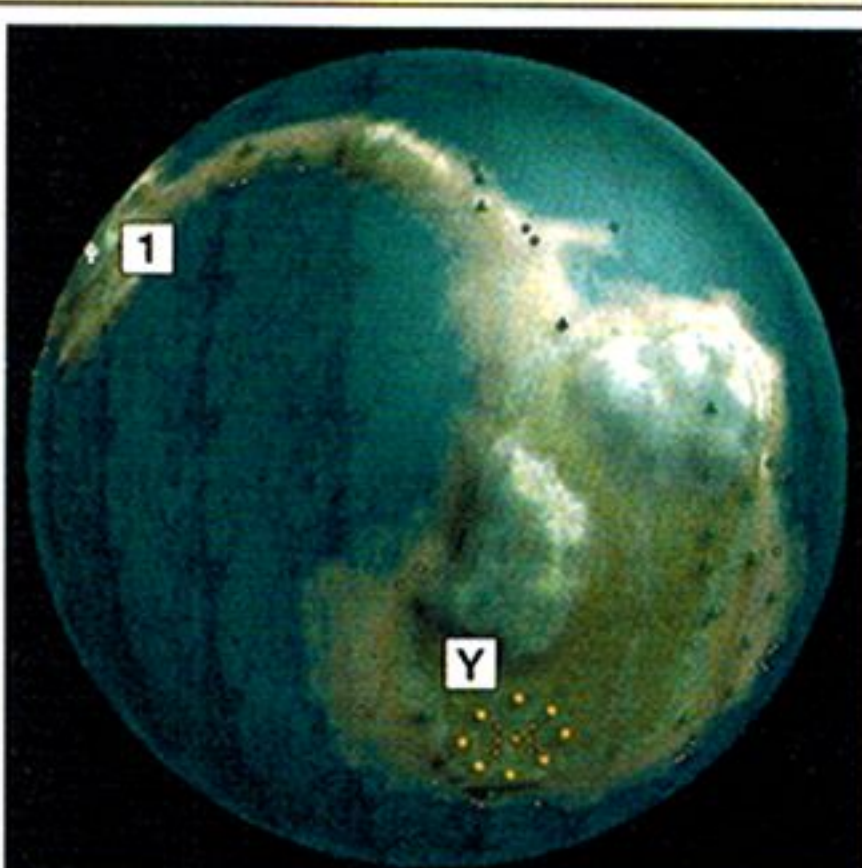
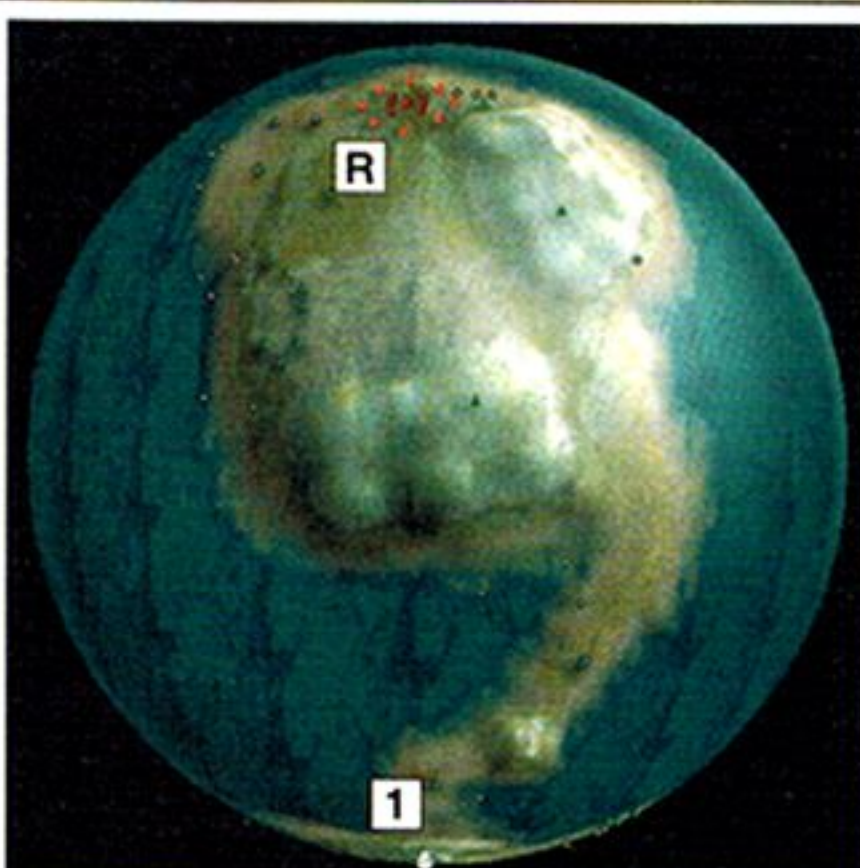
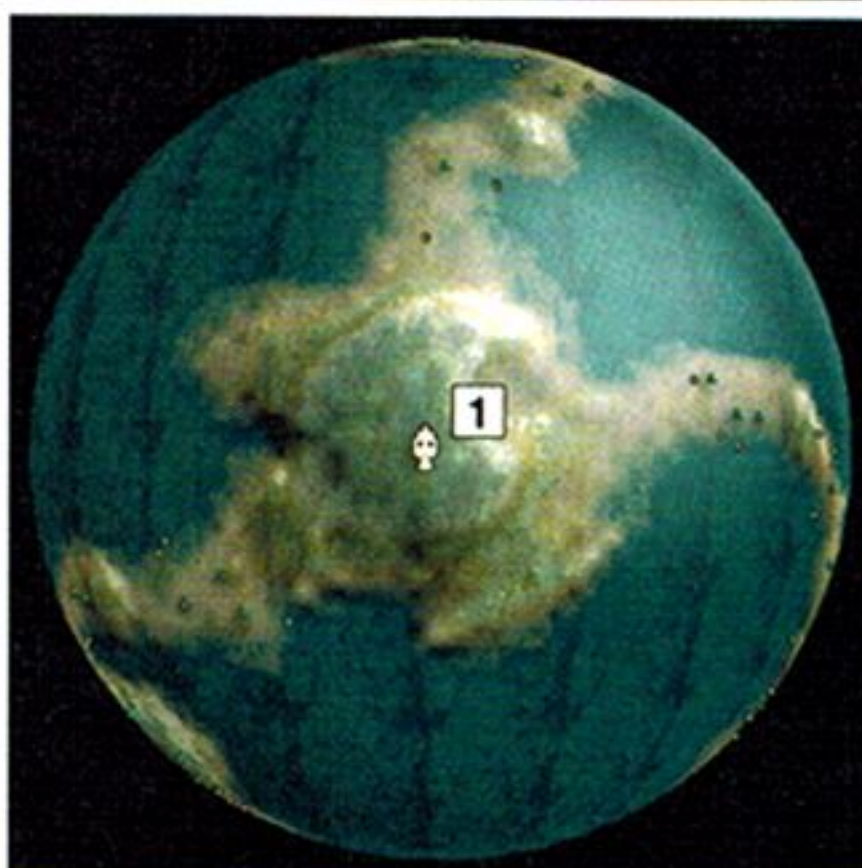
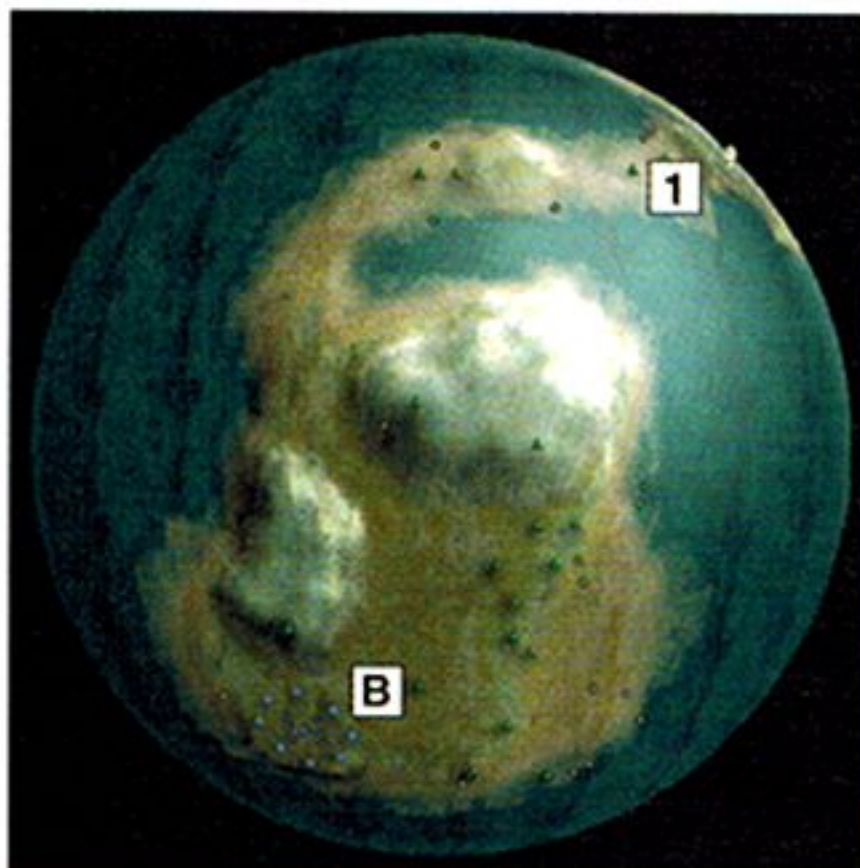
２プレイヤー版で紹介していた，敵地へ「土地隆起」スペルを4発使って陸続きにして侵攻を仕掛けてしまう「裏侵攻ルート」作戦は，プレイヤー１(青)とプレイヤー２(赤)の陣地間では可能だが，プレイヤー３(黄)はプレイヤー１(青)，プレイヤー２(赤)いずれの陣地にもこの作戦が行えない。ワールドビューマップを見てもらうとわかるが，プレイヤー３(黄)の陣地は他の2種族の陣地から遠く離れているからだ。

しかし，裏を返せば，プレイヤー３(黄)はほかの２人のプレイヤーからこうした手段で奇襲される心配がないということでもあるわけで，防衛策は陸路に対してのみ整えておけばいいということになる。

このレベルでもプレイヤー３(黄)は傍観者となる可能性があるようだ。

ガードタワーでストーンヘッドを護衛する

自軍陣地の出入り口付近の高台はこのようなガードタワーによる最終防衛ラインを作っておいたほうがいい

プレイヤー１(青)とプレイヤー２(赤)の陣地は海図上では互いに比較的近い位置関係にあるのだが……

プレイヤー３(黄)の陣地は他の２種族の陣地から遠く離れているのだ

このレベルの概略図　1.「地震」スペル(×3)

3つの道
THREE WAY

心がまえ

各自の陣地はそれぞれ独立した島として与えられる。島の形こそ違うがの地形的条件はほぼ同じ。

なお，土地があまり広くなく，木々が少ないので，建設プランはよく考えて立てる必要がある。このレベルは気球とボートが初期状態から使えるので，これを早く作れるようになることが大切だ。

なお，各自の陣地とは別に誰のものにもなっていない無人島が存在する。ここは非常に資源が豊かなので，ここを敵よりも早く開拓していくことが勝利の糸口となる。

速攻でいく

攻撃的にいくか，防衛重視でいくか，で建築戦略もかなり変わってくる。

敵が，小屋の建築に夢中になっているようであれば，いち早く，気球小屋と炎の戦士訓練小屋を作り，気球軍団を敵地に向かわせ，一気に勝負をかけるといい。

まず，シャーマンを殺し，あとは手も足も出ない勇士やその他の従者を個別に撃ち殺していけば一気に勝負がつく。

ただ，3プレイヤーゲームなので，ある敵の陣地の攻撃に夢中になっていると，自軍本拠地が別の敵に攻撃されていたりするので注意が必要だ。

防衛重視でいく

長期戦になることをにらみ，防衛重視でいく場合は，敵のボートや気球による攻撃に対処するために島の外周全体にガードタワーを適当な間隔で建てていき，そこに炎の戦士を住まわせて，全方位からの侵攻に対応できる完璧な防衛ラインを築いてしまうといい。

なお，自軍も移動手段を確保しなければならないので，ボート小屋か気球小屋も初期のうちに建てておくこと。

伝道師小屋は万が一，敵に上陸されたときのためにやはり築いておく。

小屋はとりあえずあと回しでいいだろう。人口増加は別の場所(後述)で行うからだ。

とりあえず速攻型の戦略でいかない場合は，まず自軍が，ゲーム序盤で全滅させられないように努めるのだ。

敵が侵攻してきたら

防衛ラインの構築を完了する前に敵が気球やボートで侵攻してくることもあるはずだ。

やっかいなのは速攻型戦略のところで紹介した，炎の戦士の気球軍団だ。こちらに炎の戦士がいない場合はかなり不利な戦いを強いられる。

この危機を先読みし，ゲーム開始直後から「催眠」スペルをチャージしておこう。「催眠」スペルは気球の上の敵従者にもアバウトな照準でヒットしてくれ，しかも「催眠」がかかったと同時にその敵従者は気球から飛び降りてくれるのだ。海上にいた場合はそのまま飛び込み自殺することになり，地上に降り立った場合でも，炎の戦士ならば，即座にそれまで仲間だった敵気球軍団に攻撃を仕掛けて相打ちになって死んでくれる。

このレベルに限らず，気球による侵攻が予想される場合には，「催眠」スペルは常にフルストックしておきたいものだ。

最初から使えるスペル	
地震	土地隆起
土地平坦	透明
腐食	幽霊軍隊
トルネード	コンバート
催眠	ブラスト
ライトニング	

最初から建てられる建物	
ガードポスト	炎の戦士訓練小屋
小屋	気球小屋
伝道師訓練小屋	ボート小屋
戦士訓練小屋	ガードポスト

新天地へ

このレベルには，各軍勢の陣地となっている島以外にもうひとつ島が存在する。

プレイヤー１(青)の陣地からは北東，プレイヤー２(赤)の陣地からは東，プレイヤー３(黄)の陣地からは南西の位置にある島がそれだ。

この島は非常に木々が豊富な場所で、建築スペースもかなりある。

自軍陣地は土地、木々にも恵まれないので人口増加はこの島でやることを考えたほうがいい。そのためには、ほかの軍勢よりも早くここへ移動することが必要になってくる。

ポピュラスにおいてもっとも大量に人員を輸送できるのがボートだ。ゲーム開始直後に、防衛ラインを築くのと並行してボート小屋を作り、ボートに5人の勇士をこの島に移住させよう。

新天地での戦略

適当な場所にガードタワーを建てれば、その周囲にあらゆる建物が建てられるようになる。小屋を中心に建てて人口増加に努めよう。

ところで、この新天地の開拓はどの場所から始めてもいいのだろうか。

「ワールドビュー」を見てもらうとわかるが、この島の北側にストーンヘッドがあることに気づくはずだ。これは、礼拝することで、マナを1000ポイント分チャージしてくれるのだ。このストーンヘッドを占拠してしまえば強力な攻撃スペルを敵よりも早くストックできることになる。

ただ、ストーンヘッドはどの種族も奪いにやってくるはずだ。そこで、これを独占するための下準備をするのである。

まず、この島に勇士を移動させたら、島の北側、ストーンヘッドの北側の丘の上にガードタワーを建てる。そのあと人口増加用の小屋を作りつつ、この丘の上にガードタワーを建て並べていく。そしてここに炎の戦士を住まわせるのだ。炎の戦士は本拠地から移動させてもいいし、この新天地で炎の戦士訓練小屋を建てて、この島で生まれた勇士を炎の戦士に仕立ててもいい。

これで、準備完了。あとは適当な従者をここに礼拝させよう。このストーンヘッドの背後の高い丘の上の防衛ラインが、このストーンヘッドに近づく敵を攻撃し、礼拝者を危険から守ってくれるはずだ。

視点の登録

この新天地はゲームが進んでいくうちに、本拠地並に重要な拠点となってくるはずだ。

ゲーム後半はこの新天地でなにか変わった事件がないかと、しきりに気にする必要があるわけだが、この視点移動をカーソルキーやマウス移動でやっていると時間がもったいない。

そこで[Shift]キー＋[Z]、[X]、[C]、[V]で登録できる、4つのカメラポイントのいずれかに、この新天地のよく見たい場所を登録してしまおう。視点を呼び出したいときはダイレクトに[Z]、[X]、[C]、[V]を押せばいい。

このレベルは敵陣が離ればなれになっているので、敵陣の「転生の地」あたりもこのカメラポイントに登録してしまうといいだろう。

いずれは新天地と本拠地を陸続きに

この新天地は、いずれは自軍と陸続きにしてしまうことをおすすめする。

このレベルは各敵陣が互いに離れているのでマウスやキーボードだけで視点変更していたのでは間に合わない。カメラポイントを使いこなそう

この島にいかに早く到着できるかが重要なのだ

気球軍団には「催眠」スペルを放て

ガードタワーで囲めば安心して礼拝できる

島の外周にガードタワーを建てまくろう

敵が防衛ラインを築いていないようであれば気球軍団の速攻が効果的

このレベルの概略図

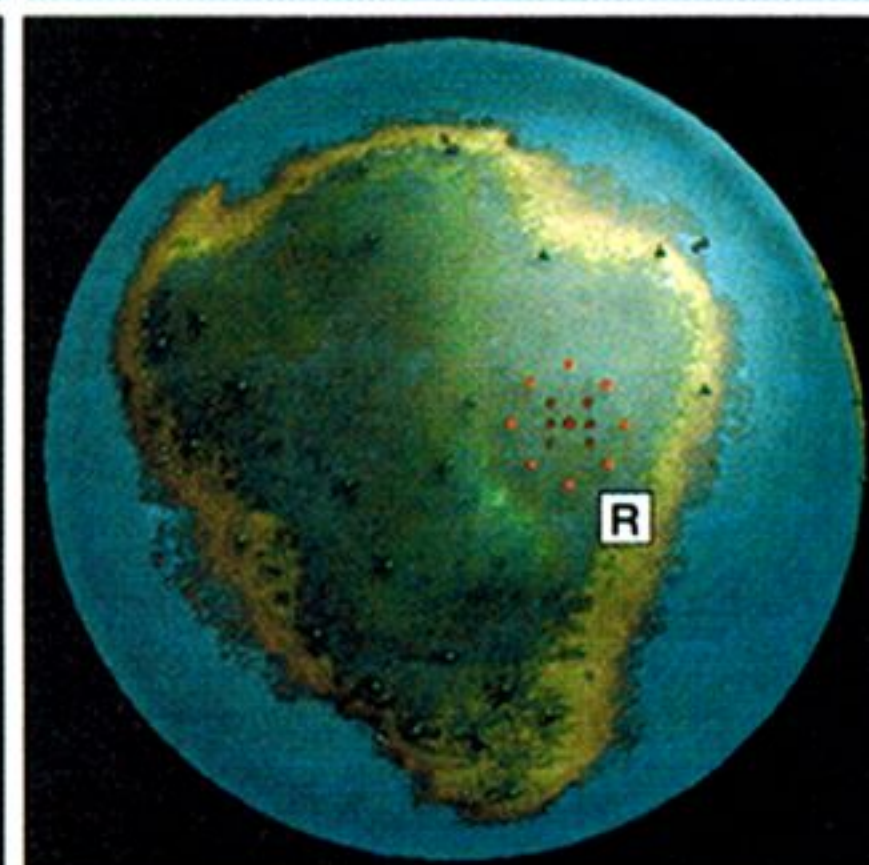
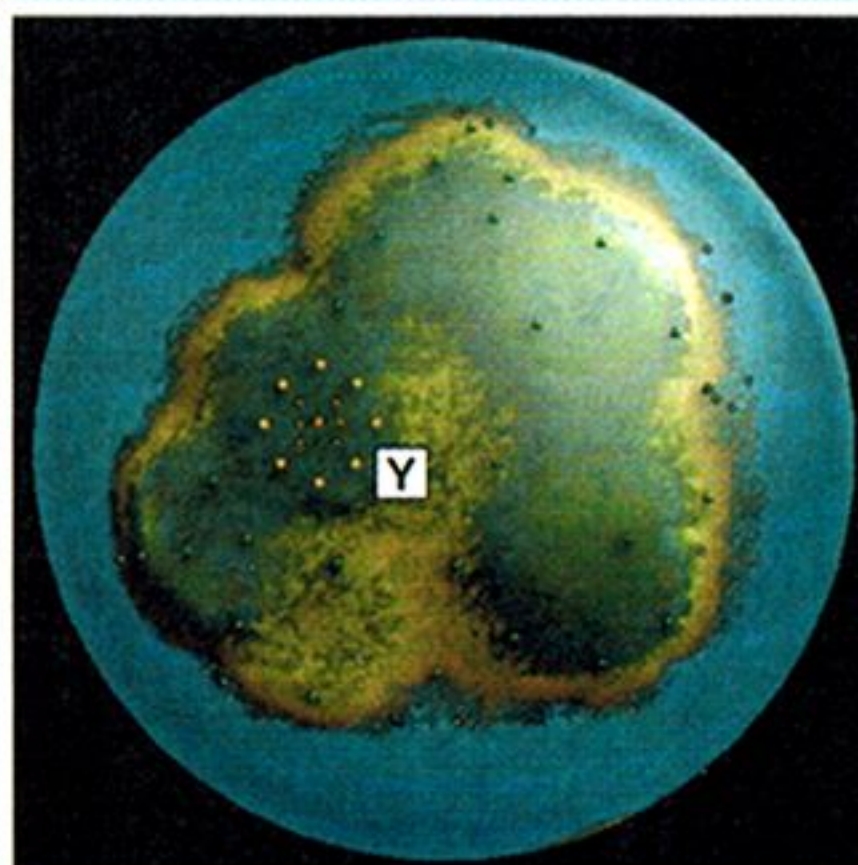
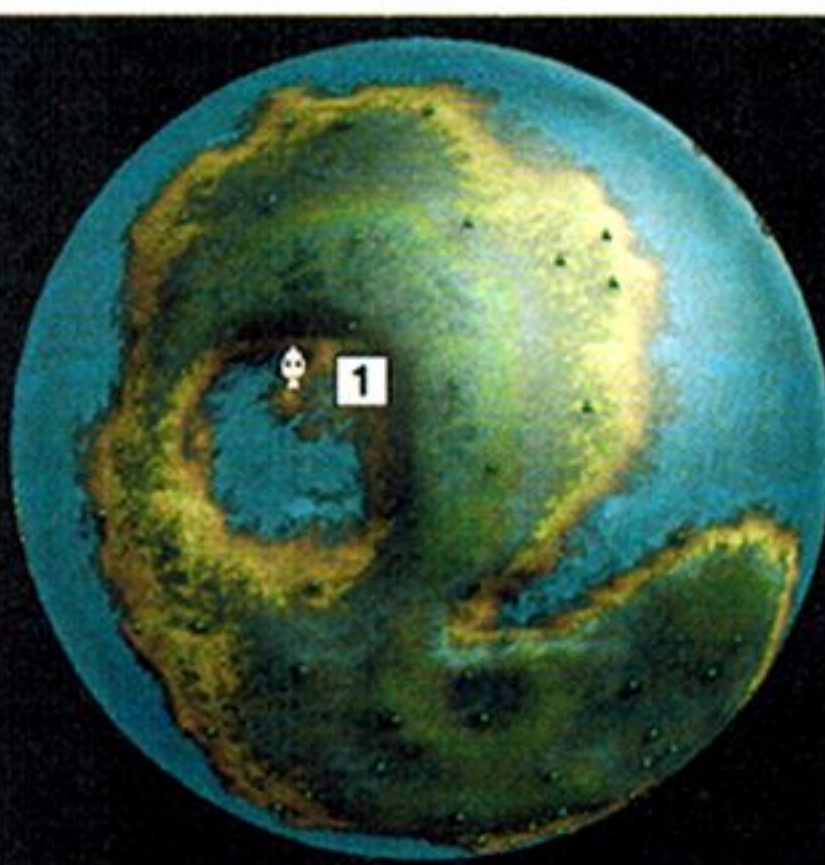
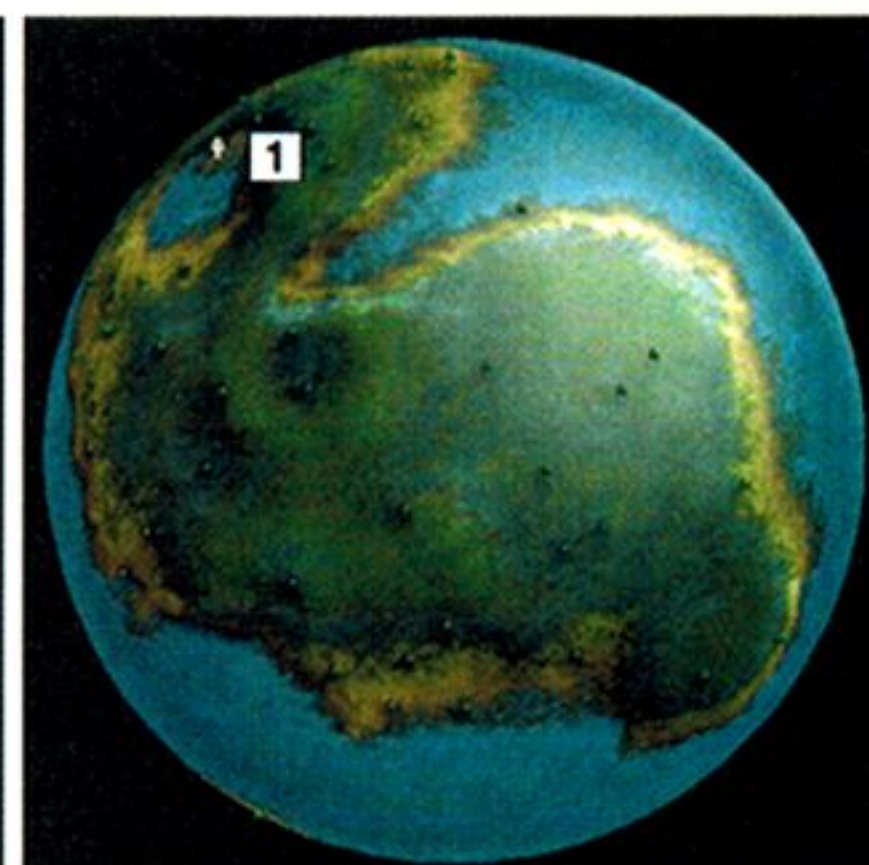
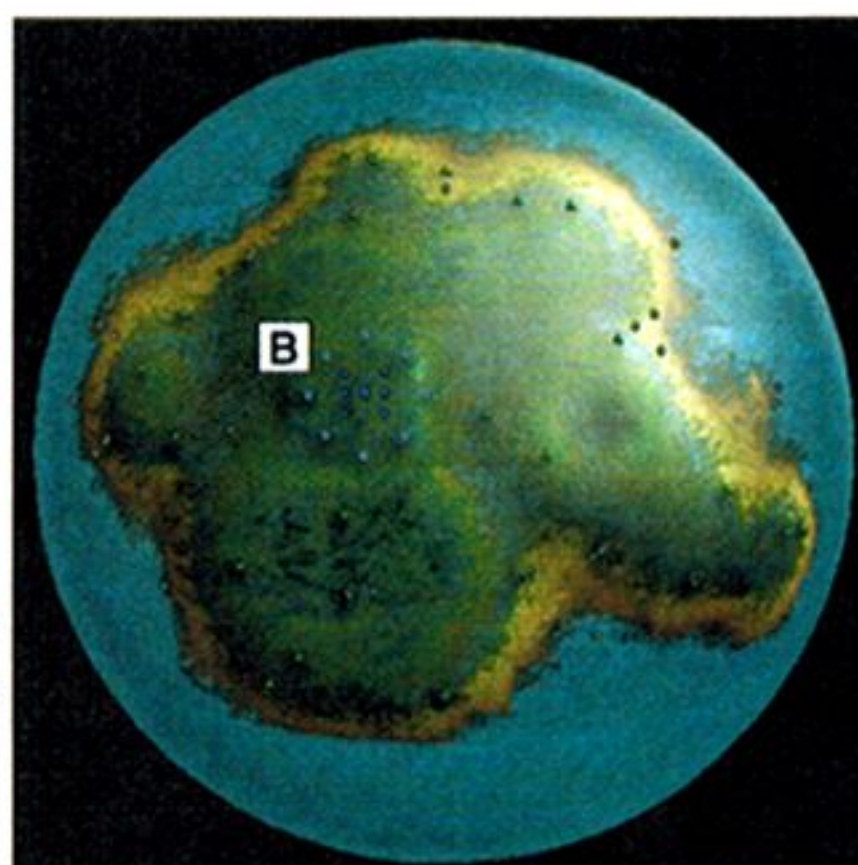
1. マナのチャージを1000ポイント分、行ってくれる(回数は無制限。有効礼拝人数は6人まで)

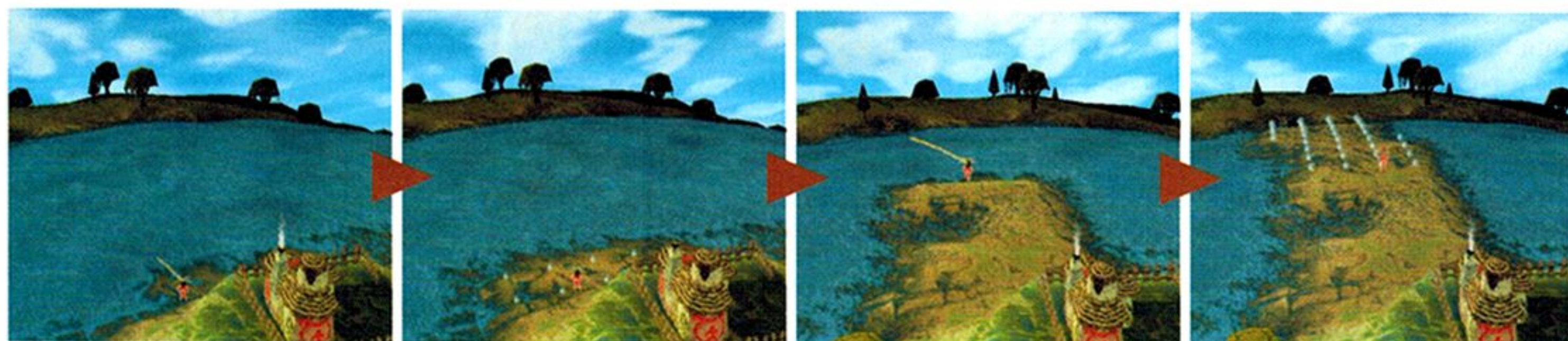
B

1

1

Y

R

それぞれプレイヤーの陣地の島には，この新天地にもっとも近いポイントが1カ所ずつ存在するので，そこから「土地隆起」スペルを使っていくといい。4～5発の「土地隆起」スペルがあれば陸続きにできるはずだ。

陸続きにできれば，新天地で生まれた勇士を，本拠地の防衛に当たらせたり，その逆も可能になる。また，本拠地，新天地いずれかが敵の激しい攻撃を受けたとしても，陸続きにしてあれば，被害を受けていないほうへ従者を逃がすこともできる。

復讐の天使
AVENGING ANGELS

心がまえ

このレベルは一見すると互いの陣地が隣接したただの3プレイヤーマップだが，実は「レベル名」に秘密があるのだ。

この秘密を利用すれば圧倒的に有利にゲームを進行させていくことができる。

その秘密の場所へ乗り物で行くか，ストーンヘッドを礼拝することで手に入る「テレポート」スペルでいくかは各自の戦略次第だ。

建設戦略

このレベルの各軍勢の陣地はほぼ同じ地形条件になっている。

ただ，平地が少ないので必要な建物から建てていく必要がある。木々はそこそこに豊富だが，野人は少ない。ゲーム序盤は計画的に物事を進行させていかなければならないだろう。

さて，自軍陣地内の平野を使い果たしてしまったら，どうすればいいのだろうか。

これは「土地平坦」スペルを使って陣地を広げていくか，あるいは，それぞれの陣地に取り囲まれた中央のフリースペースを開拓していくか，の二択になるだろう。

フリースペースに建てた建物はいうまでもなく敵の攻撃にあいやすいので，防衛ラインを築きつつ，勢力を拡大する必要がある。

防衛ライン

自軍本拠地の，そのフリースペースへ面した土地は，小高い丘になっているのに気づくはずだ。

敵は本拠地に侵攻してくる場合，この丘を駆け上がってくる必要があるわけで，これは格好の防衛ライン構築場所ということになる。

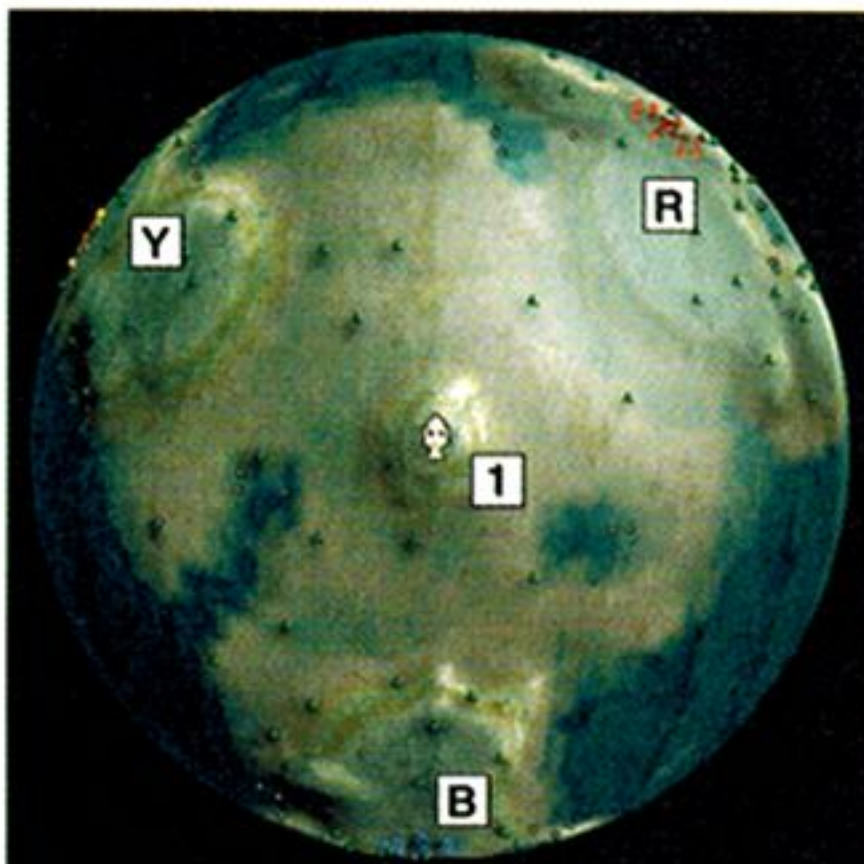
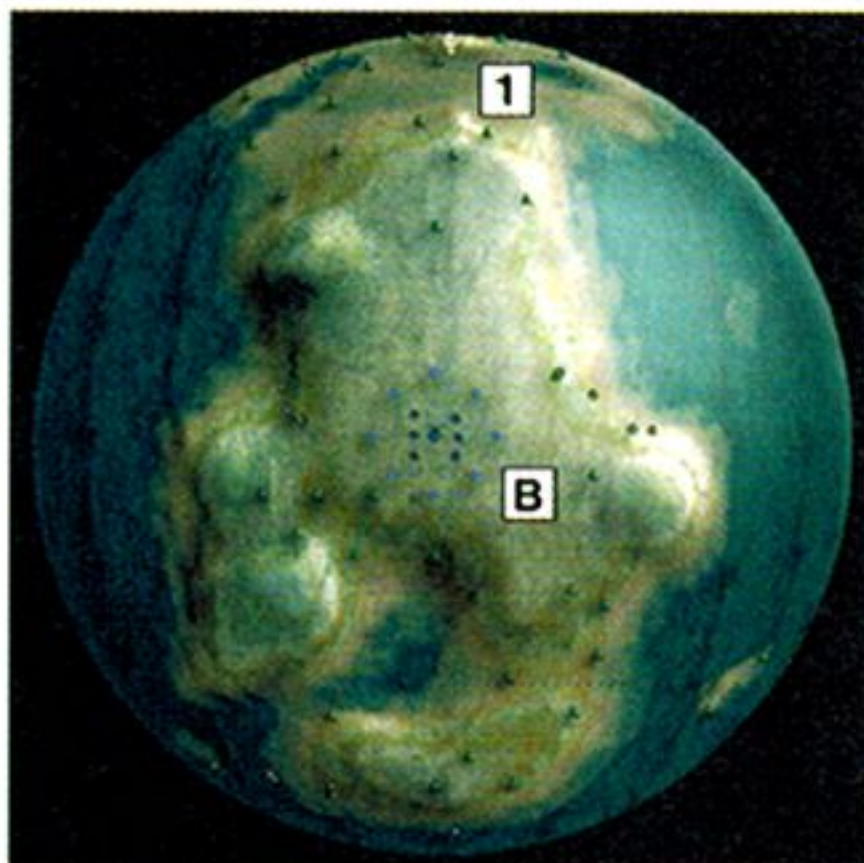
この丘に検問を設ける感覚で，ガードタワーを建て並べ，中に伝道師，炎の戦士を住まわせておくといいだろう。

ストーンヘッドはなにをくれるのか

1のストーンヘッドは実に意味ありげに存在しているが，いったいなにを与えてくれるのだろうか。これは概略図を見てもらうとわかるが，ただ「テレポート」スペルを1発与えてくれるだけなのだ。

最初から使えるスペル	
火山噴火	ライトニング
死の天使	土地隆起
ファイアストーム	マジックシールド
浸食	透明
地震	ハチの大群
土地平坦	幽霊軍隊
腐食	コンバート
トルネード	ブラスト
催眠	

最初から建てられる建物	
ガードタワー	スパイ訓練小屋
小屋	気球小屋
伝道師訓練小屋	ボート小屋
戦士訓練小屋	ガードポスト
炎の戦士訓練小屋	

このレベルの概略図
1.「テレポート」スペル（×5。ただし，礼拝に成功すると礼拝者は強制的に「ライトニング」スペルの攻撃を受けて死亡する）
2.「死の天使」を各軍勢の「転生の地」に1匹ずつ，合計3匹を一度に召還する

ストーンヘッドの周囲は平地に恵まれているので，ここにテリトリーを拡大していく。それには写真のような防衛ラインも同時に築く必要があるだろう

しかも。礼拝を完了した瞬間に「ライトニング」スペルが礼拝者を直撃し、死亡させてしまう(いうまでもないが、ここをシャーマンを礼拝させるのはやめたほうがいい)。

そんな犠牲まで払って手に入れた「テレポート」スペルはなんの役に立つのか。

実は、この「テレポート」には、「離れ小島のオベリスクを礼拝しなさい」というメッセージが込められているのだ。

離れ小島のオベリスクへ行くには

プレイヤー1(青)陣地のからは南西、プレイヤー2(赤)の陣地からは北東、プレイヤー3(黄)の陣地からは北西の位置に、離れ小島があるのに気づいただろうか。

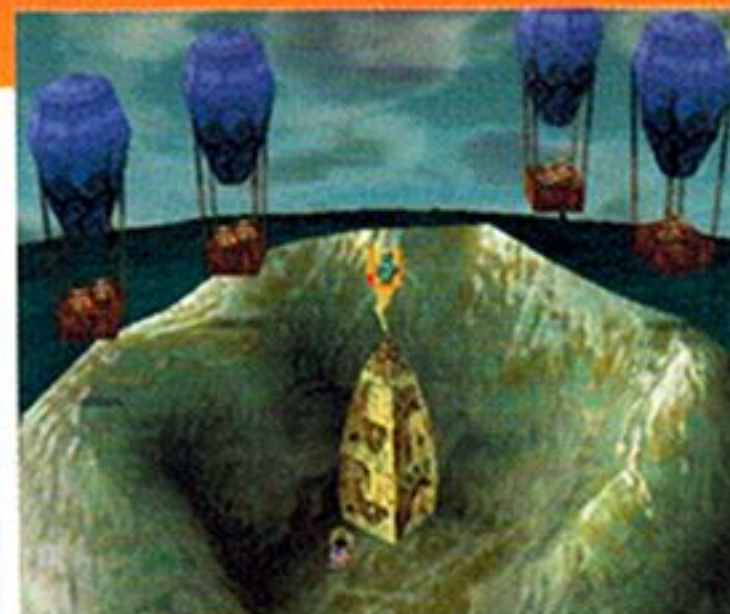
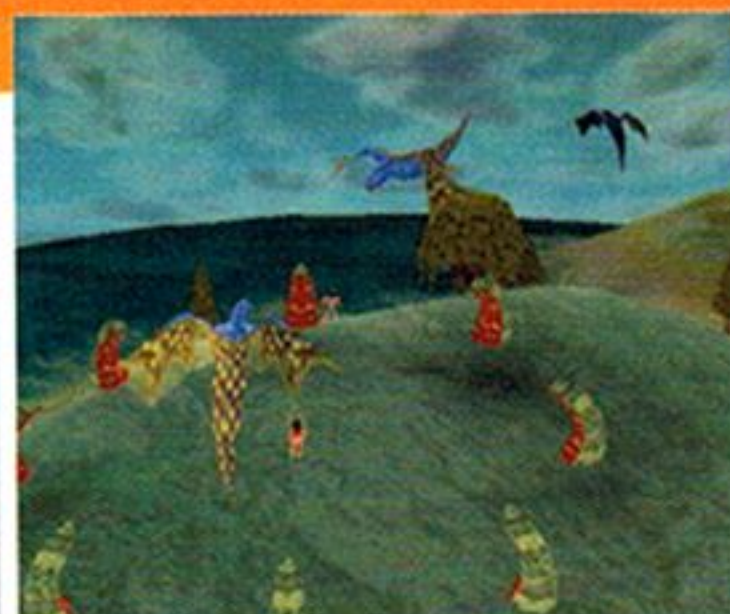
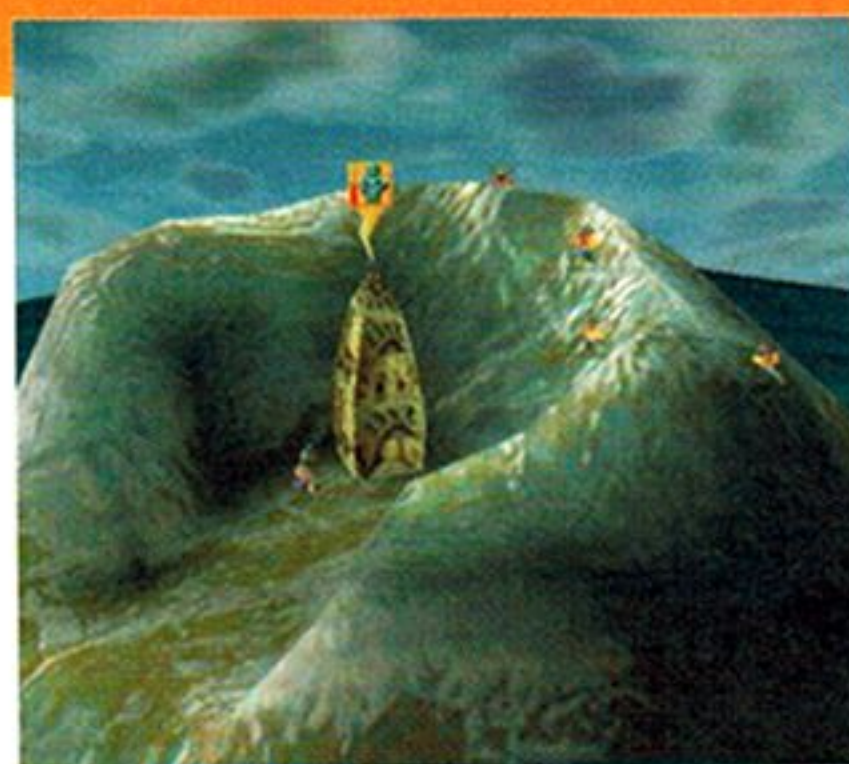
先ほどのストーンヘッドが与えてくれる「テレポート」スペルは「ここへ行け」という啓示なのだ。

しかしこのレベルは気球やボートが使えるので、乗りもので直接この島へ行ったほうが手っ取り早い。

ストーンヘッドを祈っていると敵の攻撃も受けやすいし、礼拝を完了したとしても、その礼拝者が死んでしまうというマイナス面もある。

そこで、もし、このオベリスクの礼拝を狙うならばゲーム開始早々に気球小屋やボート小屋を作り、そこで作った乗りものでこの島に急行するといい。

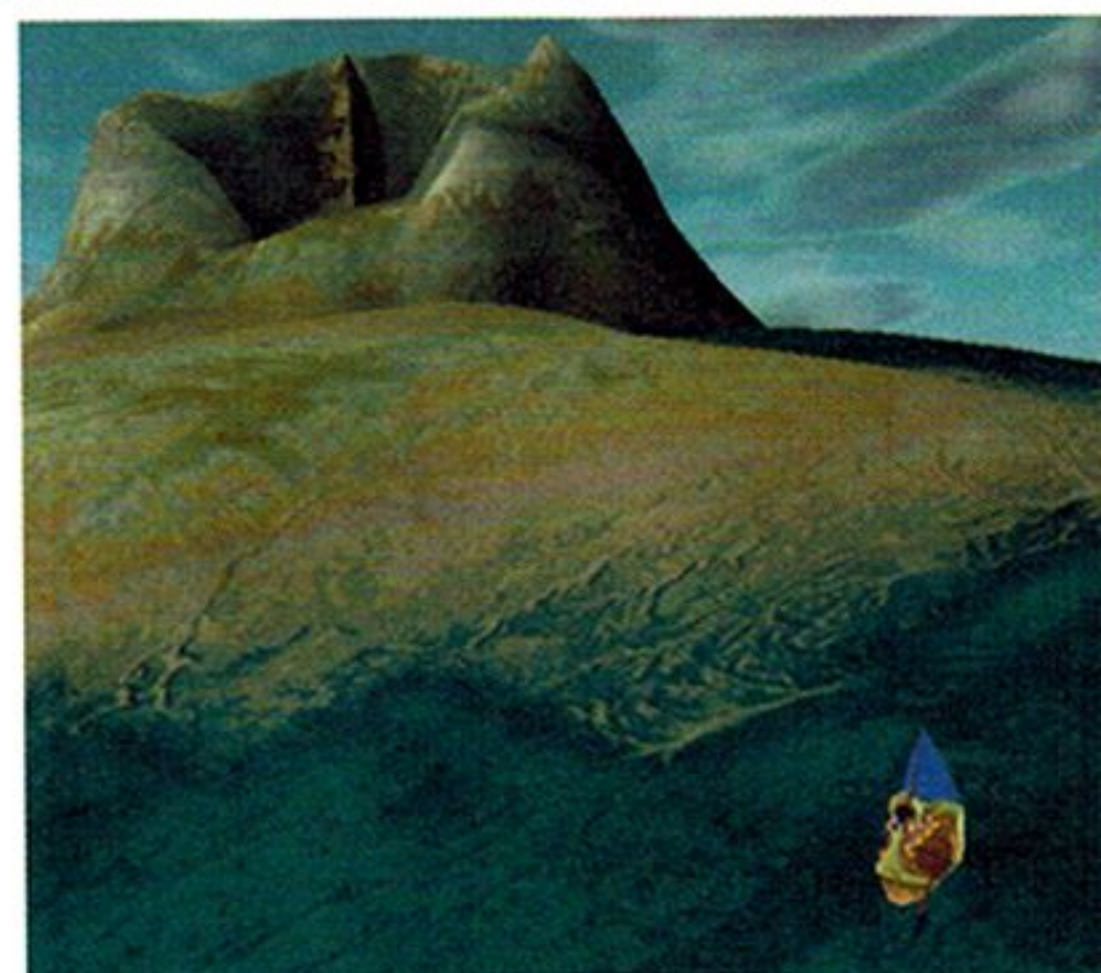
復讐の天使

さて、問題のこのオベリスクはいったいなにを与えてくれるのか。

実は、このオベリスクはなんと「死の天使」を同時に3体も召還してしまう奇跡を与えてくれるのである。

死の天使の1体あたり30人の敵従者を食い殺す能力があるので、単純計算で90人を食い殺してくれることになる。実際には移動時間や敵の抵抗があるのでそううまくはいかないとしても、相当なダメージを敵軍勢に与えてくれることは確かだ。

祈る側としてはなんとしてでもここの礼拝は成功させたい。オベリスクを礼拝できるのはシャーマンだけなのでなんとかしてこれを護衛する手段が必要になってくる。

もっとも単純なのは、ボートに礼拝者となるシャーマンのほかに炎の戦士を4人連れていき、このオベリスクの周囲にある丘に張り込みをさせるという戦法だ。敵がボートや気球で妨害しにきた場合に反撃してくれるはずだ。

気球をたくさん作っている場合には、この丘の頂上に炎の戦士の気球軍団を停泊させて防衛ラインを築くという手もある。

インチキ臭いほど強力な「死の天使」の3体同時出現。恐ろしいことにこの死の天使は各軍勢の「転生の地」の真ん中に1体ずつ出現するのだ

いずれにしても礼拝中のシャーマンは絶好の攻撃目標になってしまっているので、プレイヤーは礼拝が完了するまで、頻繁にこの周囲に目を配る必要があるだろう。

砂の城
SANDY CASTLES

最初から使えるスペル

ファイアストーム	土地隆起
浸食	透明
地震	ハチの大群
腐食	幽霊軍隊
催眠	コンバート
ライトニング	ブラスト

最初から建てられる建物

ガードタワー	戦士訓練小屋
小屋	炎の戦士訓練小屋
伝道師訓練小屋	ガードポスト

心がまえ

各軍は比較的広い台地を陣地として与えられ、ここを本拠地として勢力を拡大していくことになる。

陣地はそこそこ広いのだが木々が少ない。よって建設は計画的に進める必要があるだろう。

各陣地に囲まれた平地の中央には巨大な池があり、ここに野人が密生している。ゲーム開始直後は、自軍陣地内の野人獲得は後回しにしてこの池の周りの野人獲得に集中したほうがいいようだ。

なお、このレベルは非常に戦いが長引く傾向にある。「ハルマゲドン」などのゲストスペルをオンにしたほうがいいだろう。

防衛ライン

各軍が陸続きで互いに向かいあっているレベルなので、たいていの敵はある程度の兵力が整うと直線的に正面(具体的にはプレイヤー1(青)の陣地では北側から、プレイヤー2(赤)の陣地では西側から、プレイヤー3(黄)の陣地では東側から)から攻めてくる傾向にある。

陣地の敵地に面している側はそこそこに高さのある崖になっているのでガードタワーを建てて防衛ラインを構築するには最適な場所となっている。ここにガードタワーを建てて主に炎の戦士、いくつかに伝道師を住まわせて、正面からの侵攻をシャットアウトしよう。

テリトリーの拡大

「復讐の天使」レベルでもそうだったが、各軍勢の陣地に取り囲まれた中央のフリースペースは平地で建築に向いているので、ここまでテリトリーを拡大することも視野に入れる。その場合はそこにも防衛ラインを形成していくこと。

危険を冒して、この中央の地に進出などしたくないというのであれば、敵地に面している側とは反対側、海側の平地に、陣地を拡大していくのもいいだろう。

背後からの侵攻・その1

敵地へ侵攻する際には、なにも馬鹿正直に正面から切り込む必要はない。

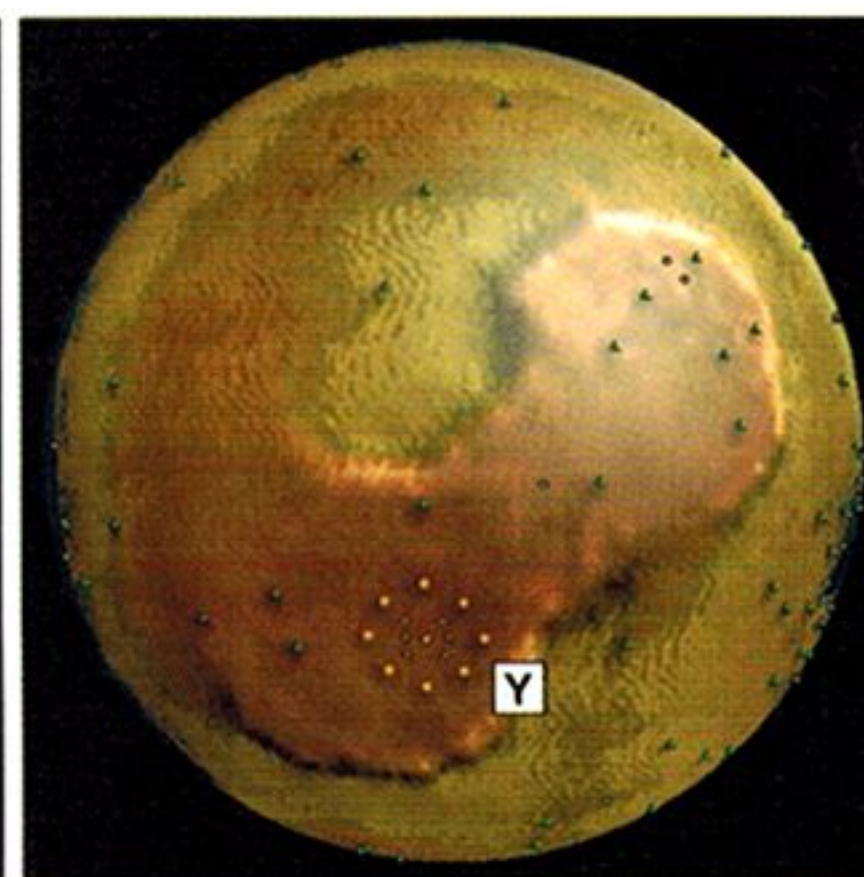
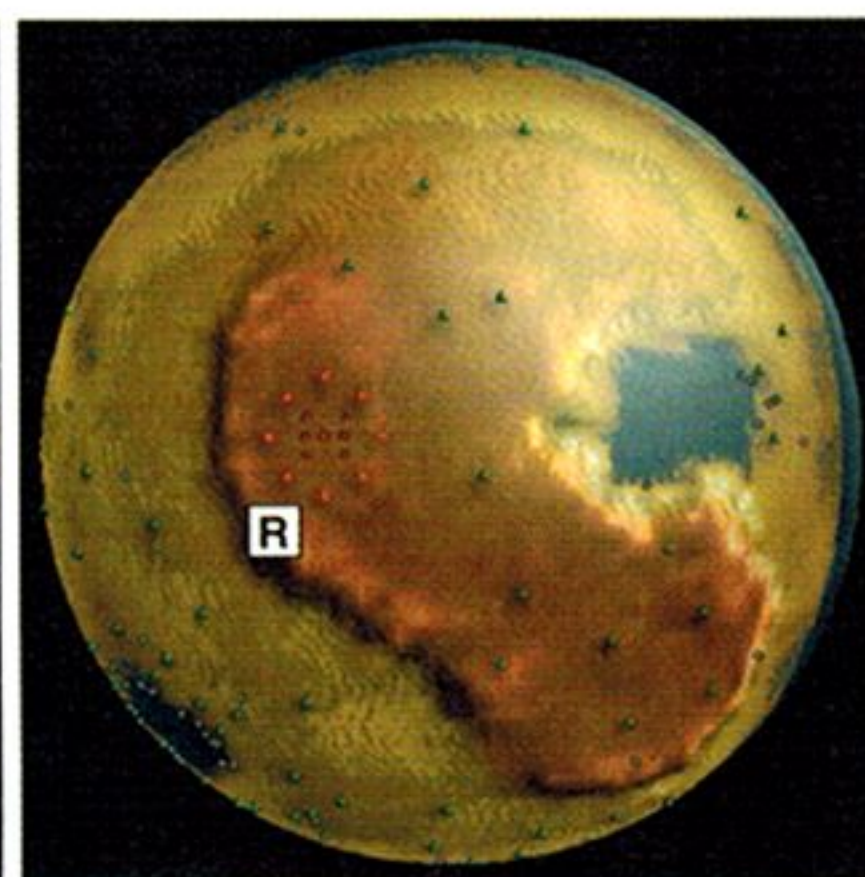
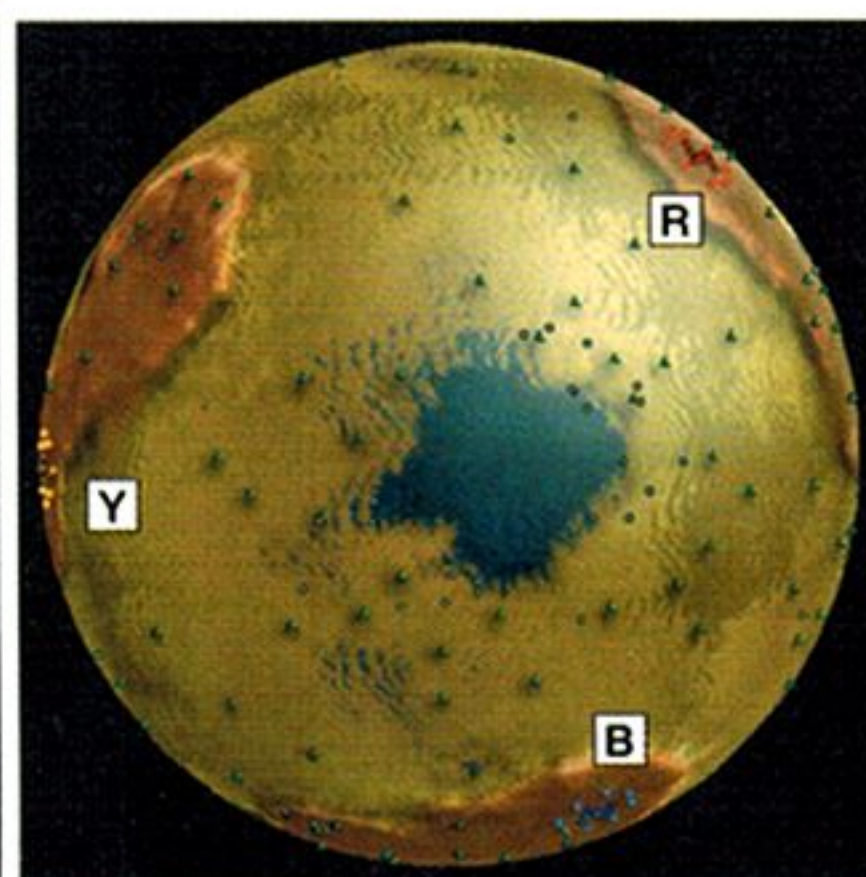
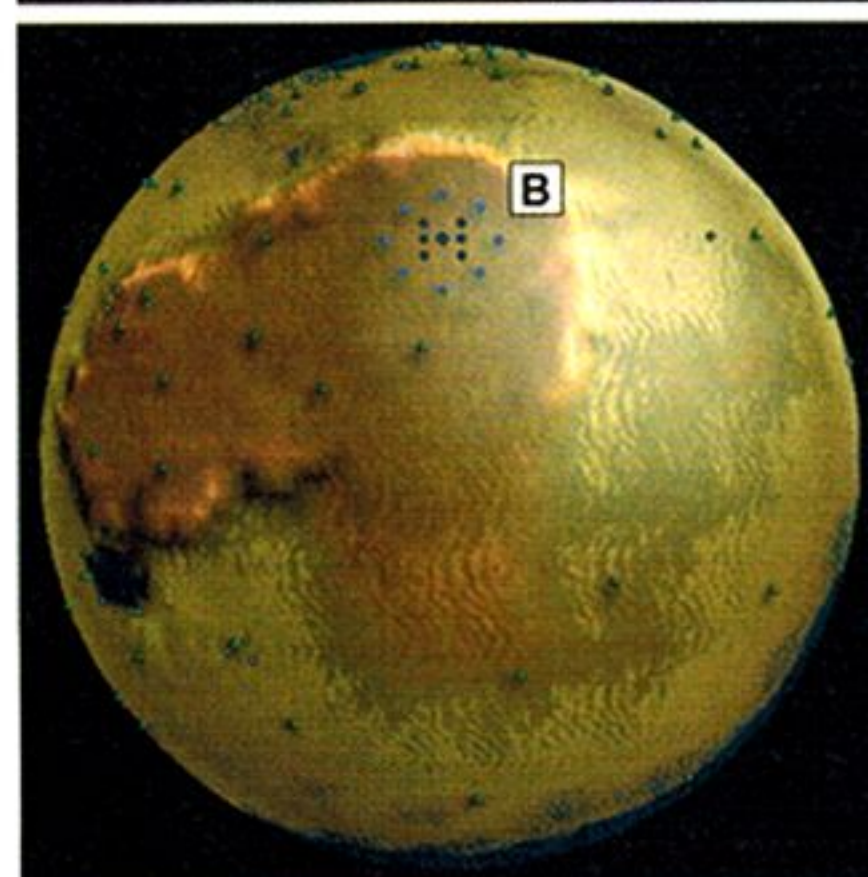
各陣地の背後(具体的にはプレイヤー1(青)は南側，プレイヤー2(赤)は東側，プレイヤー3(黄)は西側)は，崖状の正面側と違ってなだらかで登りやすい斜面になっているので，ここからのほうが侵攻がしやすいのだ。

ただ，軍勢を裏から進軍させたのでは，バレバレなので，少し工夫する必要がある。

単純ながら効果的なのは陽動作戦だ。

まず，「幽霊軍隊」スペルを駆使し，「それらしい」ニセの軍勢を作り，先にこれを敵地正面側に集結させ，アピールする。

敵プレイヤーの意識がそちらに集中している間に，その敵陣の背後から攻め込むのだ。侵攻はニセ部隊と

このレベルの概略図

本物部隊を同時に仕掛けると面白いだろう。敵はかなり混乱するはずだ。

背後からの侵攻・その2

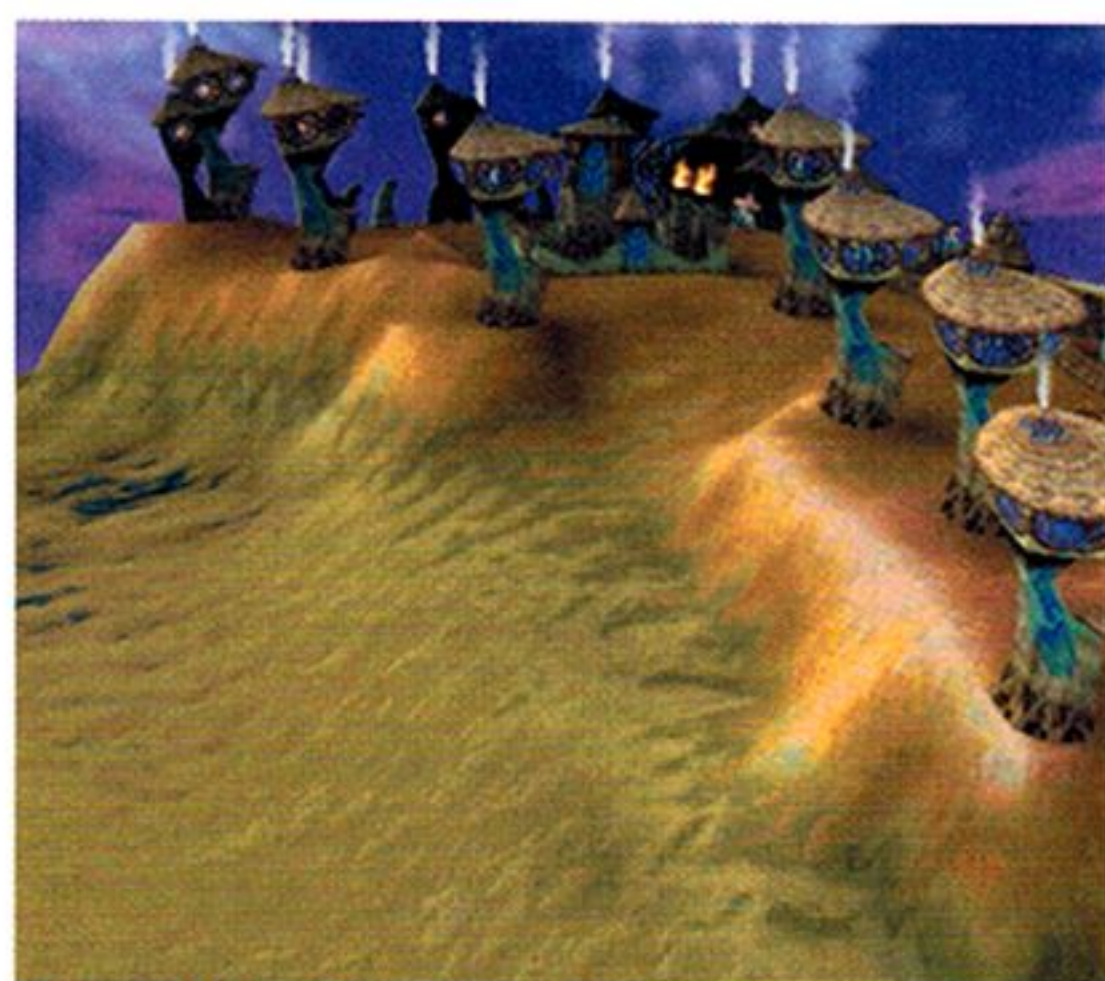
この戦法をさらに応用してこんな戦法も考えられる。ニセ部隊を見せつけて，敵プレイヤーにアピールしている間に，本物部隊には「透明」スペルをかけてしまい，これを裏から侵攻させるのだ。透明で侵攻させるのは伝道師がメインになるだろうが，戦士も透明にして送り込み，敵陣の防衛ラインのガードタワーを破壊させるのも作戦としては有効だろう。

裏侵攻ルート

敵にカウンターアタックをかます意味あいも含めておすすめしたいのが，「土地隆起」スペルを使っての奇襲作戦だ。

「ワールドビュー」を見てほしい。

各陣地は正面は互いに陸続きではあるが，その裏側を見ると海を挟んで隣接しているのである。そしてそれは，場所によっては「土地隆起」スペル1発で陸続きになってしまうような，かなり近いところもあるのだ。

ここをあえて陸続きにしてしまい，敵に侵攻を仕掛けるのが，その奇襲作戦だ。

この「裏」侵攻ルートとなりうるポイントはこの2つ。

- プレイヤー1(青)陣地の南側は，プレイヤー2(赤)陣地の北側，プレイヤー3(黄)の北東側と隣接している
- プレイヤー3(黄)陣地の北西側は，プレイヤー2(赤)陣地の東側に隣接している

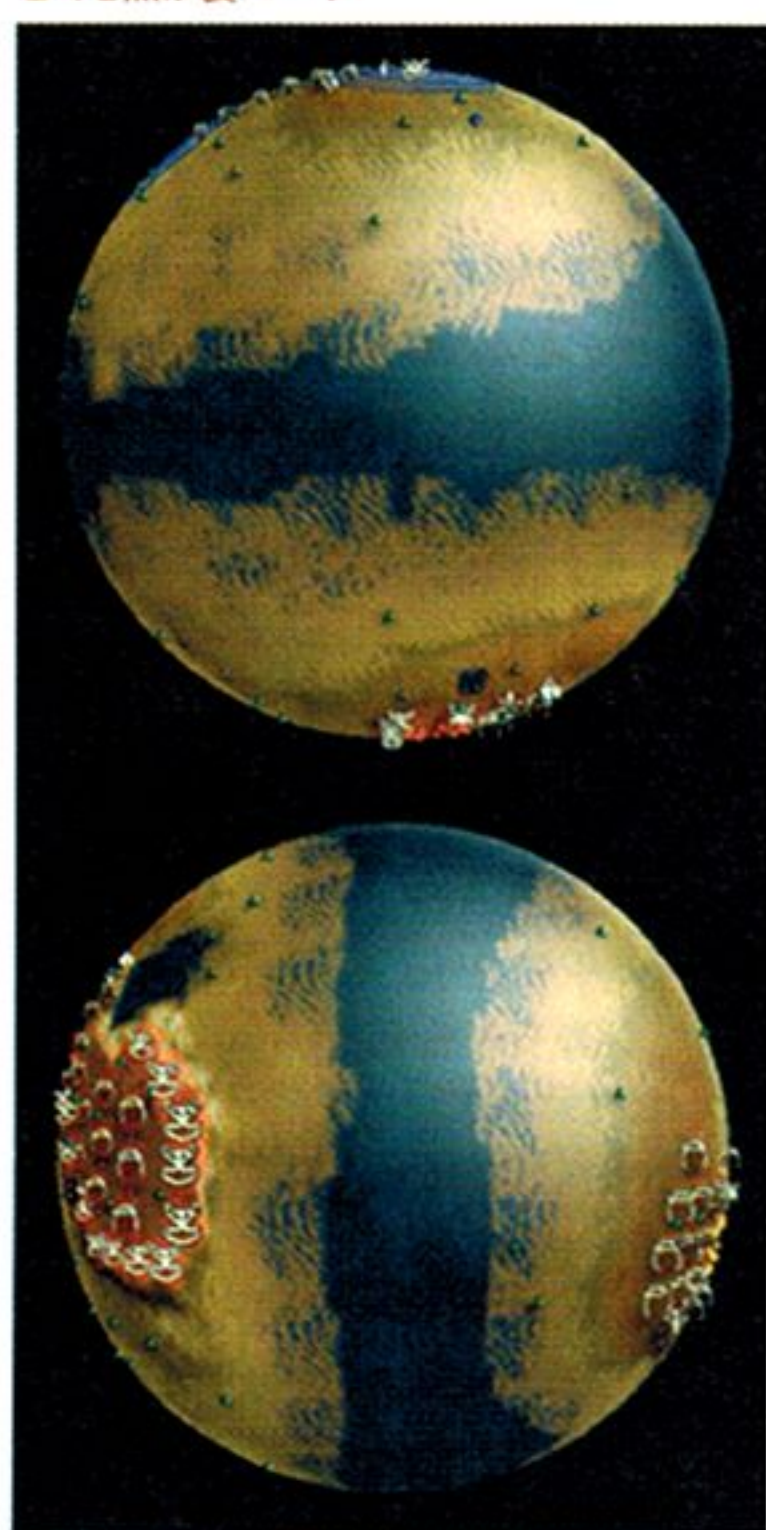
さて，いずれのポイントも双方の陣地の背後側同士を接続できるものなので，陣地正面ばかりに気を取られ，陣地背後の防衛策をおろそかにしていたプレイヤーは，この戦法で足をすくわれることになるわけだ。

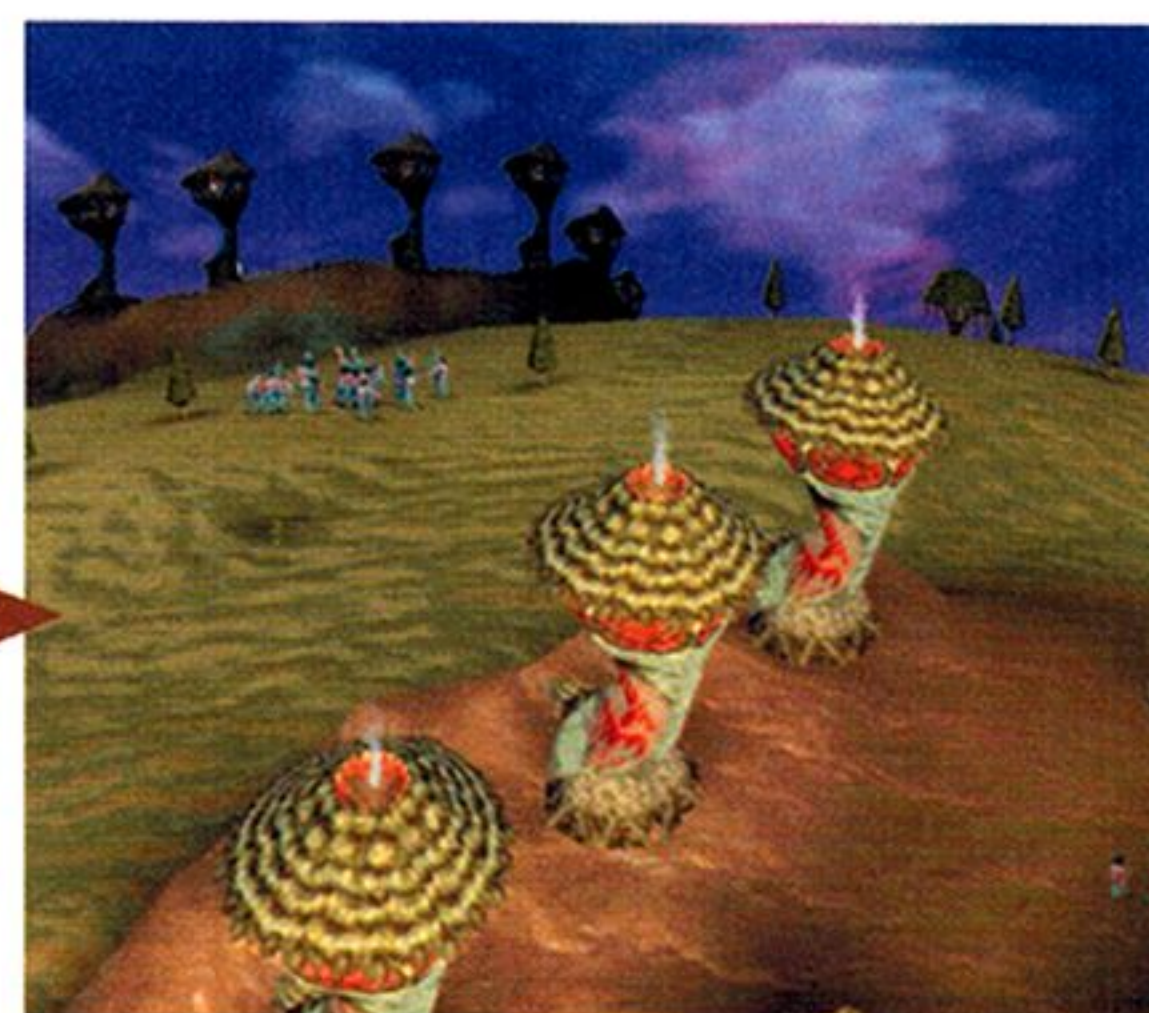
守る側としてはこれらの裏侵攻ルートをあらかじめ予測し，陣地の背後側も強力な防衛ラインを構築する必要があるということになる。

さて，このレベルはあまりにも土地が広大なためか，従者に移動命令を出すと，ときどきとんちんかんな方向に歩き出してしまうことがある。これは各従者のルート決定アルゴリズムが単純な直線最短距離を歩こうとするものだからなのだが，この陸路ができると，それまでとんちんかんなルートで移動しようとしていた従者たちも，この陸路を積極的に使うようになり，スピーディな移動ができるようになる。そういう意味でこの陸路を作るのもいいかもしれない。

高い位置のガードタワーには炎の戦士，進入経路になりそうな緩やかな斜面には伝道師を住まわせておくといい

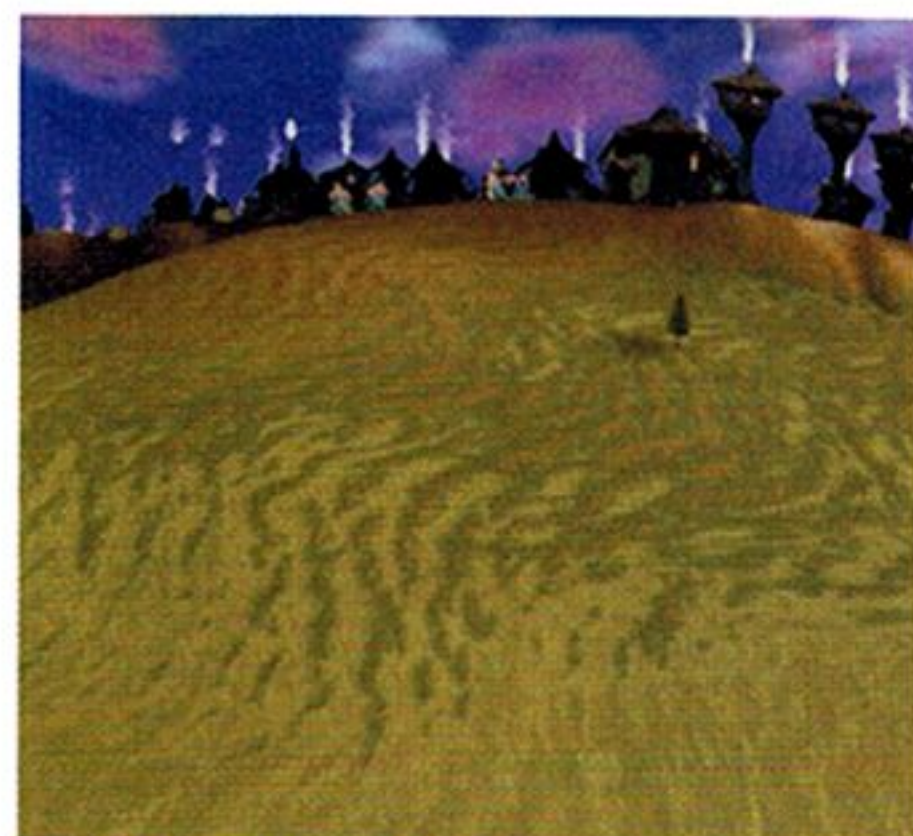
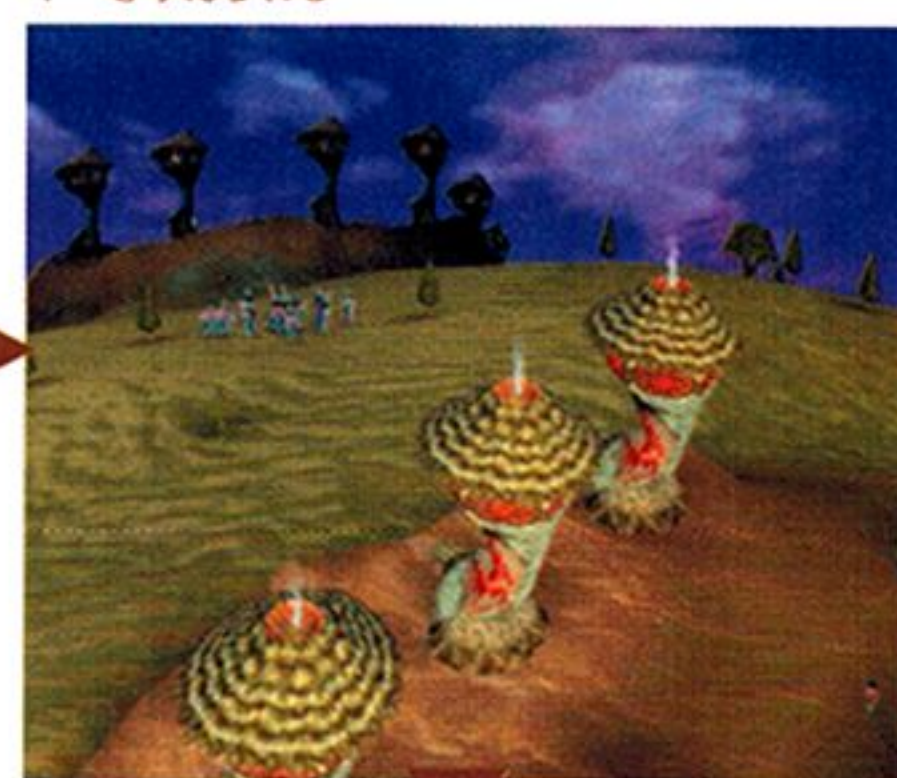
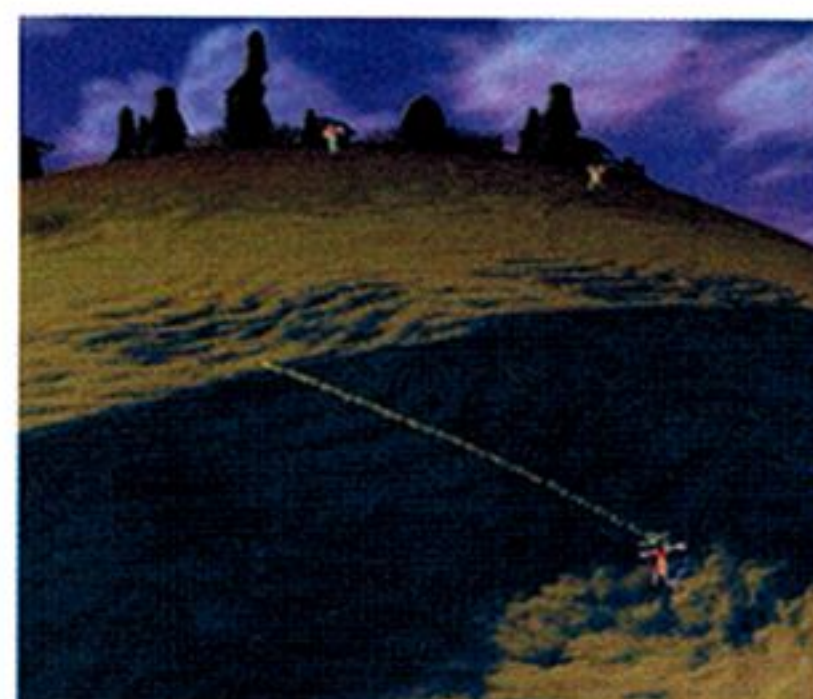
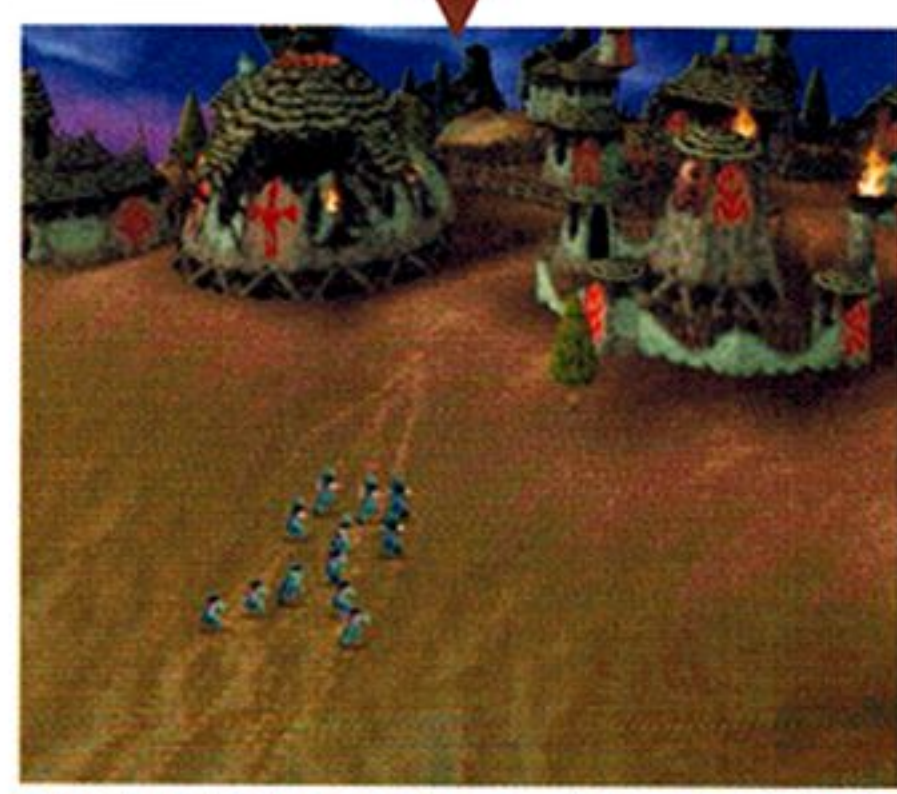
正面側は鉄壁の防衛ラインを築きたい

この2点が裏ルート

本物の侵攻部隊に「透明」スペルをかけて……

ニセ部隊を侵攻させたりして，敵の目を欺きながら，

背後から透明の本物の侵攻部隊を敵地に侵入させてしまう

戦士や伝道師に「幽霊軍隊」スペルをかければその偽物を作ることができる

「幽霊軍隊」スペルによる偽物部隊も，

敵の目からはこう見えるわけで，かなりのプレッシャーを与えられる

陣地の背後には広い平野が広がっている

このように「土地隆起」スペル一発で陸続きにできる

囮に意識を集中させておいて，本物の侵攻部隊は敵陣地の背後から侵入する

4プレイヤーマップの完全攻略

ツーオンツー
TWO ON TWO

最初から使えるスペル	
ファイアストーム	土地隆起
地震	マジックシールド
土地平坦	透明
腐食	ハチの大群
トルネード	幽霊軍隊
催眠	コンバート
ライトニング	ブラスト

最初から建てられる建物	
ガードタワー	スパイ訓練小屋
小屋	気球小屋
伝道師訓練小屋	ボート小屋
戦士訓練小屋	ガードポスト
炎の戦士訓練小屋	

心がまえ

4人対戦用のマップでありながら、プレイヤー1(青)とプレイヤー2(赤)、プレイヤー3(黄)とプレイヤー4(緑)がそれぞれ陸続きの同じ島に陣地を与えられるため、同盟の組み方によって遊び方も変わってくる。その組み合わせは以下のような感じだ。

・**全員が敵対するする場合**
(同盟設定が全員〈中立〉となっている場合)
プレイヤー1(青)対プレイヤー2(赤)、プレイヤー3(黄)対プレイヤー4(緑)という構図になりやすく、その場合、それぞれの勝者がのちに戦うことになる。

・**プレイヤー1(青)とプレイヤー2(赤)が同盟、プレイヤー3(黄)とプレイヤー4(緑)が同盟を組んだ場合**
互いの陣地で共生しながら、敵対するチームの島へ侵攻していく。

・**プレイヤー1(青)とプレイヤー3(黄)が同盟、プレイヤー2(赤)とプレイヤー4(緑)が同盟を組んだ場合**
全員敵対した場合と同様のゲーム進行になりやすい。

ゲーム開始時にはこれらの位置関係を踏まえた上で同盟関係を設定したほうがいいだろう。

同じ島に敵がいる場合

中央の池を挟んでほぼ対称の島で、同じ島に敵がいる場合はそれぞれの軍勢が南北に分かれて戦うことになる。

この場合、このレベルは「2組の非常に小さな2プレイヤーマップ」ととらえていいだろう。

こうして考えた場合、互いの陣地の距離は2プレイヤー用レベルの「運命の丘」よりも短く、さらに陸続きなので、速攻が有効となる。

伝道師小屋を先に建て、伝道師を仕立てこれをどんどん敵地に送り込んでやろう。「マジックシールド」スペルをこれにかけて送り出せるとなお心強い。

速攻を仕掛けないのならば、近い将来の敵の侵攻に備えて防衛ラインを構築したほうがいい。

構築場所は互いの陣地の中間にある池の近くのやや高くなっている丘の上がいい。ガードタワーを数本建てて炎の戦士、伝道師を住まわせる。

敵が、この丘に無関心ならば、池の向こう側、敵地側の丘の方にも防衛ラインを作ってしまうのも面白い。敵はこちらに侵攻を仕掛けるためには2段構えの防衛ラインを突破する必要に迫られるため、かなり優位な立場に立つことができる。

敵が海の向こうの島にいる場合

同じ島にいる種族が味方プレイヤーなのであれば、戦略はかなり変わってくる。敵同士の陣地が離れているので、侵攻はボートか気球に乗って行う必要があるのだ。そこで、ゲーム開始後、早々に、気球やボート小屋の建設を行おう。

防衛ラインは、島中央の小高い丘ではなく、島の外周に形成したほうがいい。特に、上陸されやすいなだらかな土地には重点的に構築すること。防衛ラインは、ガードタワーに炎の戦士を住まわせたものでいいだろう。伝道師はこの防衛ラインのガードタワーに住まわせるのではなく、万が一上陸されたときのための対応人員として、陣地中央に待機させておく。

敵の防衛ラインを破壊するには、シャーマン自らが気球に乗って出かけていき、これを「ライトニング」スペルで焼き払うか、あるいは「マジックシールド」スペルをかけた炎の戦士の気球軍団で、これを攻撃させるかすればいい。

もし、敵がこの戦法でガードタワーを破壊しにやってきたら、「催眠」スペルで撃退する。何度もいってきたが、気球に乗った「マジックシールド」つきの敵従者はこの方法でしか撃退できないからだ。そういう意味でも、防衛戦略上、「催眠」スペルは常にストックしておきたいところだ。

北側ストーンヘッドを先に奪え

概略図に示したように、北側のストーンヘッドは「死の天使」スペルを無制限に与えてくれる。これはぜひ自軍が奪いたいし、絶対、敵に奪わせてはならない。自軍の陣地と同じくらい、死守すべきものといえるだろう。

さて、この北側のストーンヘッドは離れ小島状の高い崖の上にあるため、ここを先取りするには、乗りものを作る必要がある。

ただし、ボートをこの島の海岸に直づけして上陸しても、ストーンヘッドが絶壁の上にあるため、島の海岸から登ることができない。よって、気球で行き、崖の頂上に直接降り立つか、あるいは礼拝メンバーにシャーマンを加えてボートで赴き、「土地隆起」スペルをこの島の崖の頂上に放ち、斜面を作り出してここを上がるか……とにかく、なんらかの工夫をしなければならない。

「土地隆起」スペルで斜面を作れば、ストーンヘッドが要求する6名の礼拝者を2隻のボートで運んでしまえるわけだが、その分、敵にもこのストーンヘッドへ手軽に到達する手段を与えてしまうことにもなる。

そこでおすすめしたいのは、気球軍団による礼拝だ。

まず、炎の戦士の気球軍団を12名(気球6個)程度で編成し、これをストーンヘッドに向かわせる。このうち6名をストーンヘッドに祈らせ、あとの6名(3個の気球)をストーンヘッドのある崖の外周に配置し、警戒にあたらせるのだ。

崖の縁に浮かばせた気球は非常に高度が高くなるため、その上の炎の戦士のブラスト砲撃の射程距離は飛躍的に伸びる。これにより、ボートでこの島に近づこうとした敵従者はいうまでもなく、気球できた敵従者に対しても先制攻撃を仕掛けられる。

敵にこの戦法を使われたときは、多少危険だが、シャーマンを気球でこの島に近づかせ、礼拝者及び護衛の気球軍団に「催眠」スペルをかけてしまうといい。仲間割れが起こり、礼拝がめちゃくちゃになるはずだ。「催眠」スペルのかかった従者が大勢生き残った場合は、継続して礼拝してしまい、「死の天使」スペルを横取りしてしまう手も考えられる。

ところで、南側のストーンヘッドは「テレポート」スペルを与えてくれるだけなので、危険を冒してまで礼拝する必要はない。

互いの陣地の中央付近に構築した防衛ライン

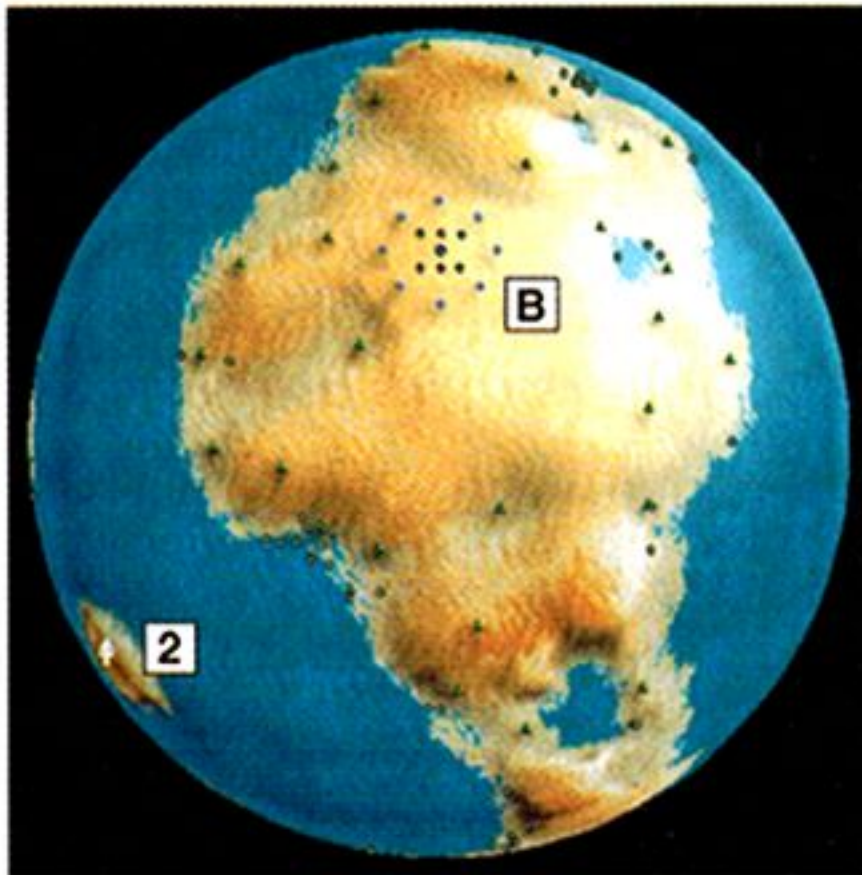
海岸線には時間をかけて完璧な防衛ラインを作ろう

「マジックシールド」スペルをかけた炎の戦士の気球軍団も防衛ライン破壊にはもってこいだ

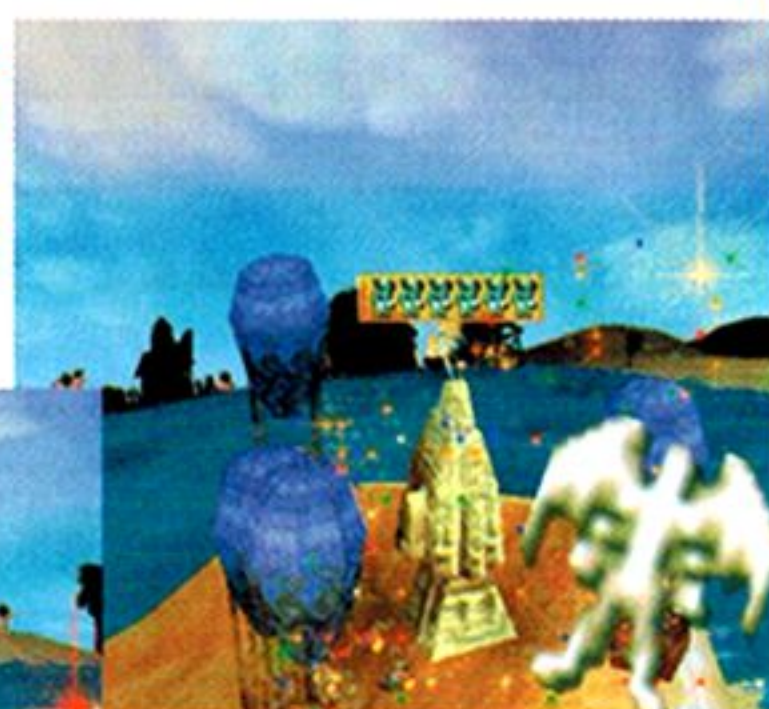
向こう岸の丘がフリーならばそちらにも防衛ラインを作ってしまおう

防衛ライン破壊において一番手堅いのがシャーマンによる「ライトニング」攻撃

攻め込まれている側は「催眠」スペルで撃退するしかない

このレベルの概略図
1.「テレポート」スペル（×5）
2.「死の天使」スペル（回数無制限）

島は絶壁になっているため，ボートで行っても頂上のストーンヘッドに行くことができない

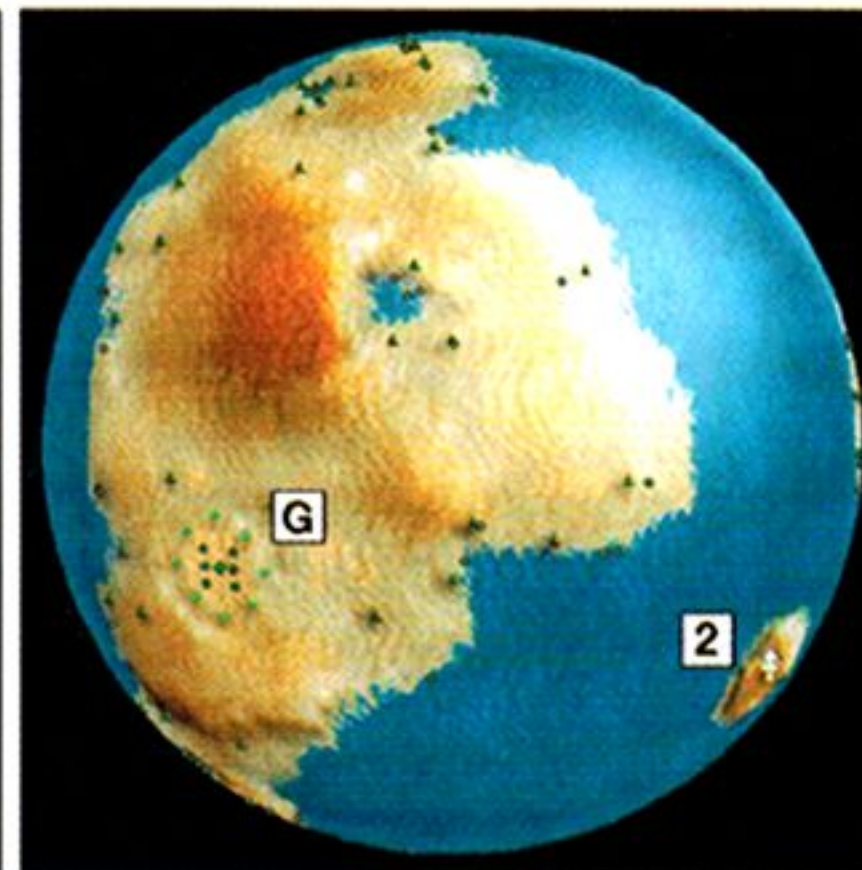
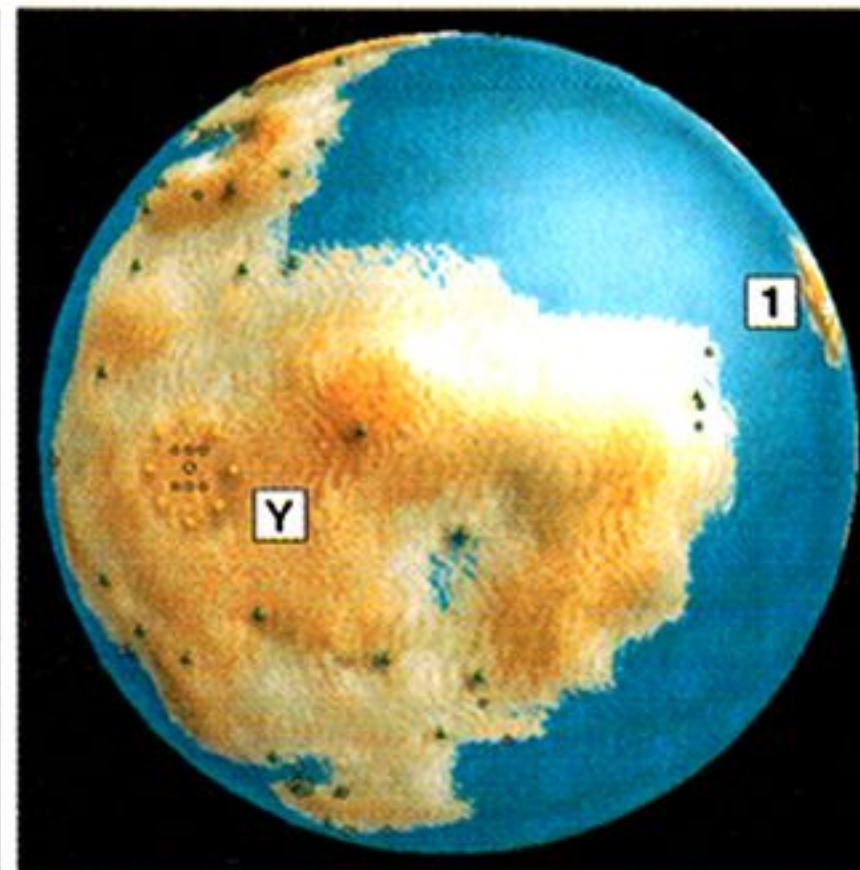
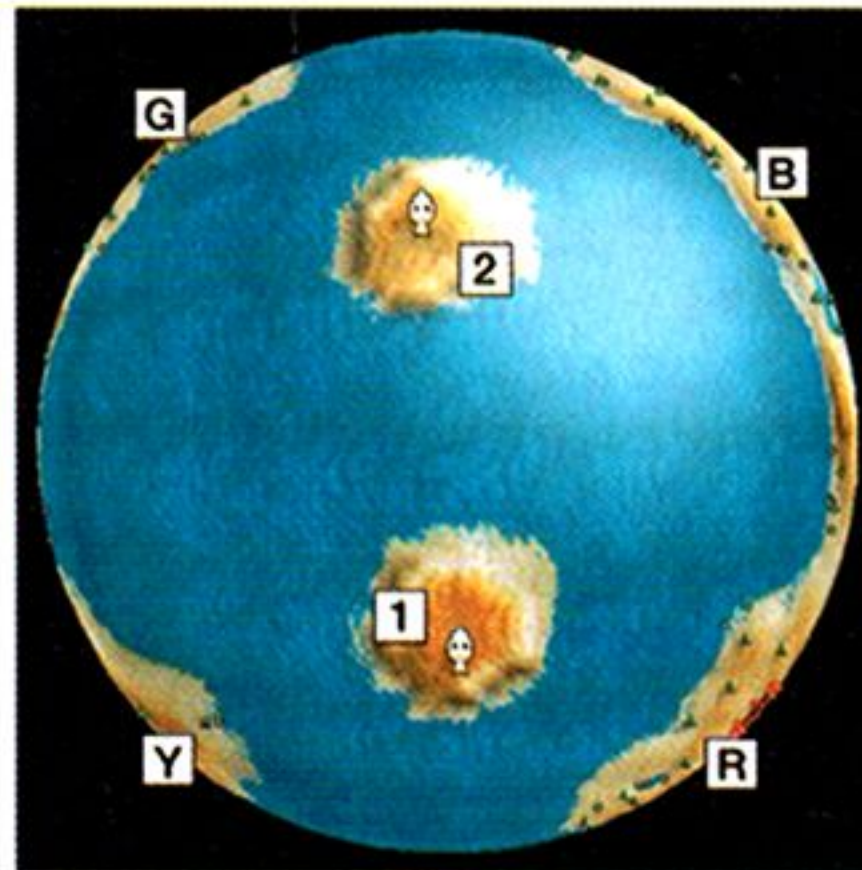
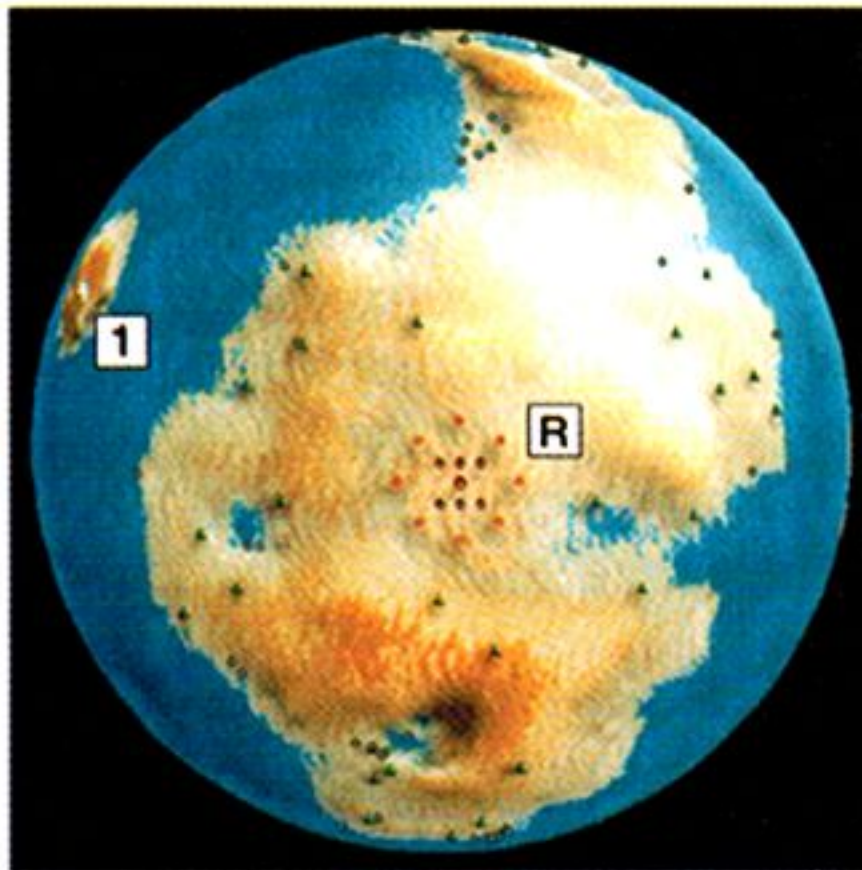
気球軍団の護衛をどんどん追加していくのもひとつの策だ。このストーンヘッドは，そこまでしてでも独占する価値があるのだ

このストーンヘッドは「死の天使」スペルを無制限に与え続けるがストックは１個だけなので，ストックされたらさっさと発動してしまうこと

南側のストーンヘッドは苦労して長い時間礼拝しても得られるのは「テレポート」スペル。割にあわない

クレーター

CRATERS

最初から使えるスペル	
火山噴火	ライトニング
死の天使	土地隆起
ファイアストーム	マジックシールド
浸食	透明
地震	ハチの大群
土地平坦	幽霊軍隊
腐食	コンバート
トルネード	ブラスト
催眠	

最初から建てられる建物	
ガードタワー	スパイ訓練小屋
小屋	気球小屋
伝道師訓練小屋	ボート小屋
戦士訓練小屋	ガードポスト
炎の戦士訓練小屋	

心がまえ

マルチプレイヤーゲームのマップとしてはストーンヘッドが多い。手にはいるのは「浸食」スペルと「ファイアストーム」スペル，「火山噴火」スペルだ。

各陣地の敷地は非常に狭いので，このどれかでも先に敵に取られて自軍陣地に投じられるかなり危機的な状況になる。

普段は使いどころのない「浸食」スペルも，このレベルでは強力な攻撃スペルとなるはずだ。

防衛ラインの早期構築と，ストーンヘッドの確保を両立させることが，このレベルで勝つための条件だ。

建築戦略

このレベルは非常に土地が狭い。野人も少ないため，獲得できる勇士の数も自ずと少なくなる。建築スペースも限られており，木々も多くない。建物は行き当たりばったりで建てていくと確実に行き詰まってしまうだろう。

まず，なにから建てていくかだが，土地が狭いので，敷地面積を多くとる炎の戦士訓練小屋，伝道師訓練小屋，気球小屋は早めに建てておいたほうがいい。

防衛ラインは島の外周にガードタワーを建てて炎の戦士を住まわせる例のパターンでいいのだが，木々が少ないので，ガードタワーの個数はあまり多くできない。適当に幅をあけて建てていき，ときどき中を埋めるような形で追加していくといい。

スペルはなにをチャージするか

「土地隆起」スペル，「土地平坦」スペルは常にチャージをオンにしておきたい。この２つのスペルはもっぱら自軍陣地の敷地拡張に使うことになる。

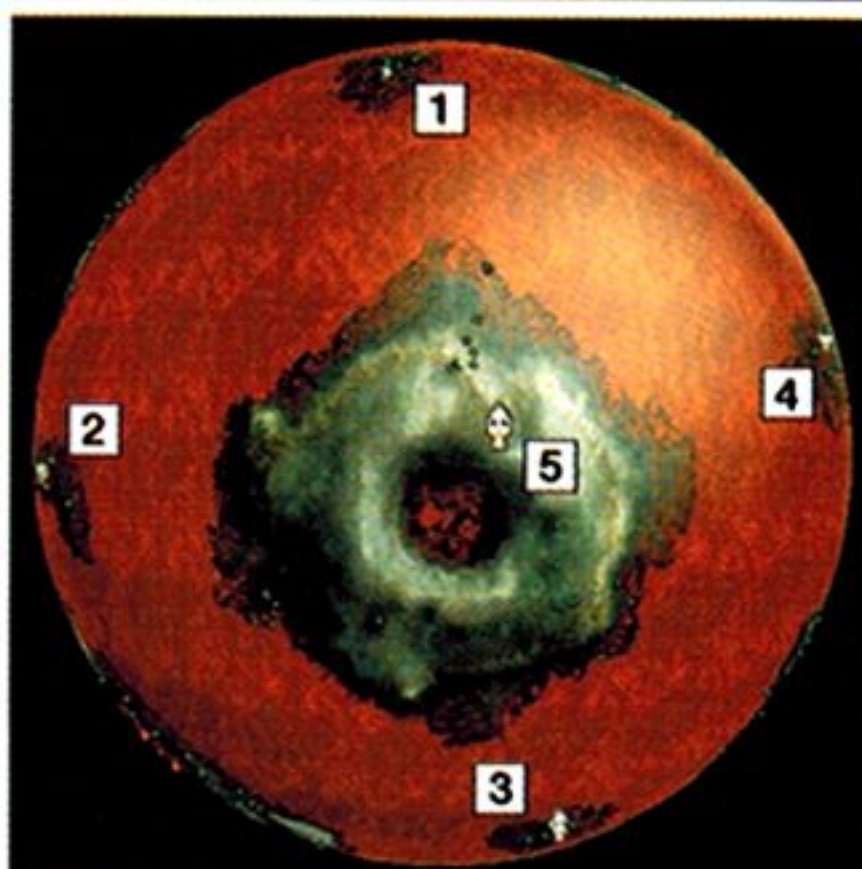
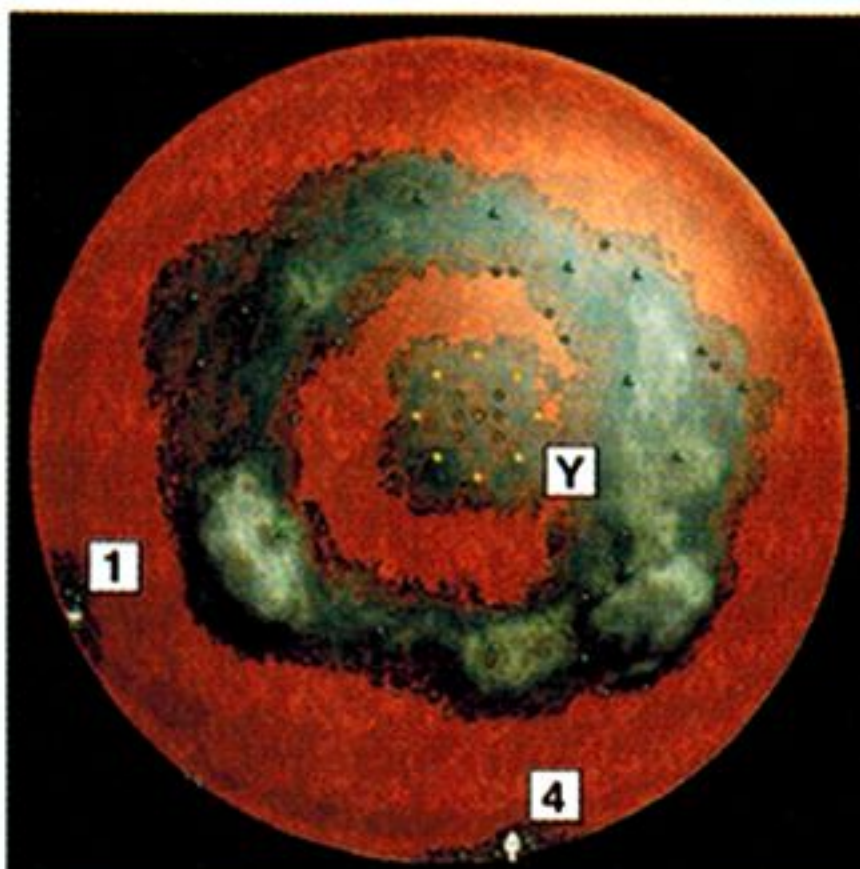
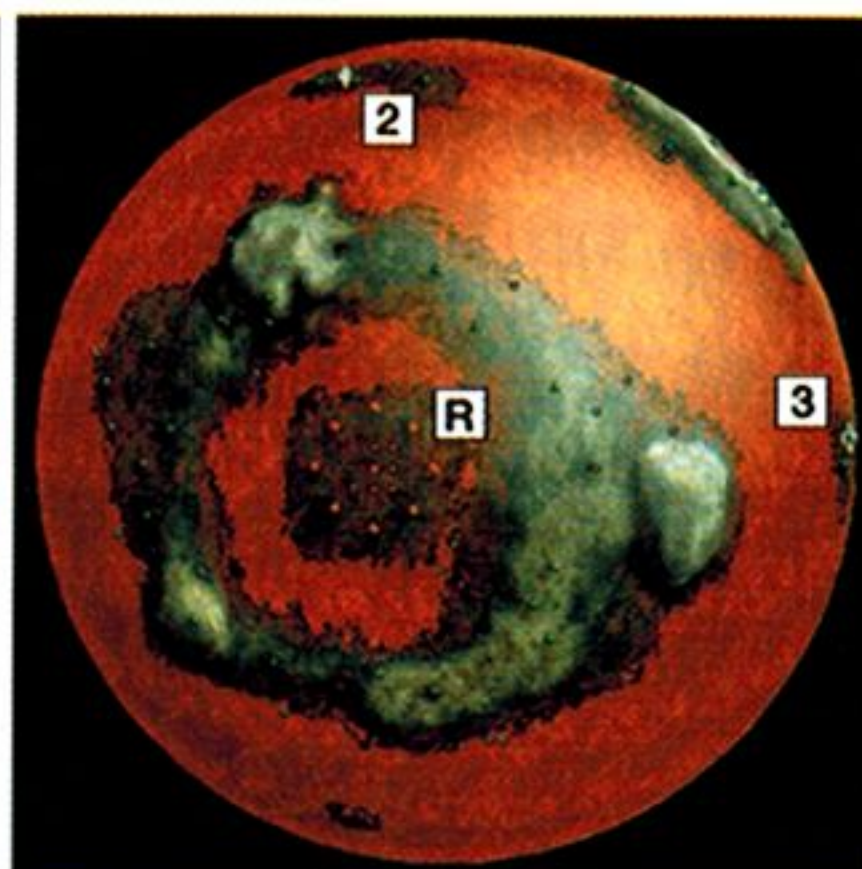
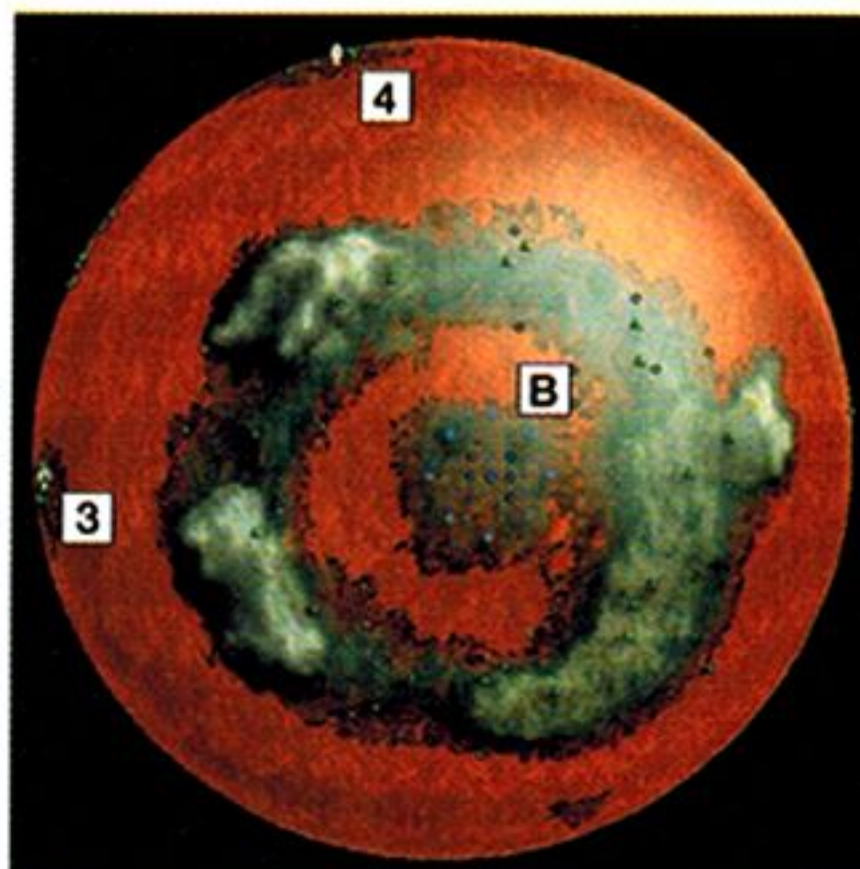
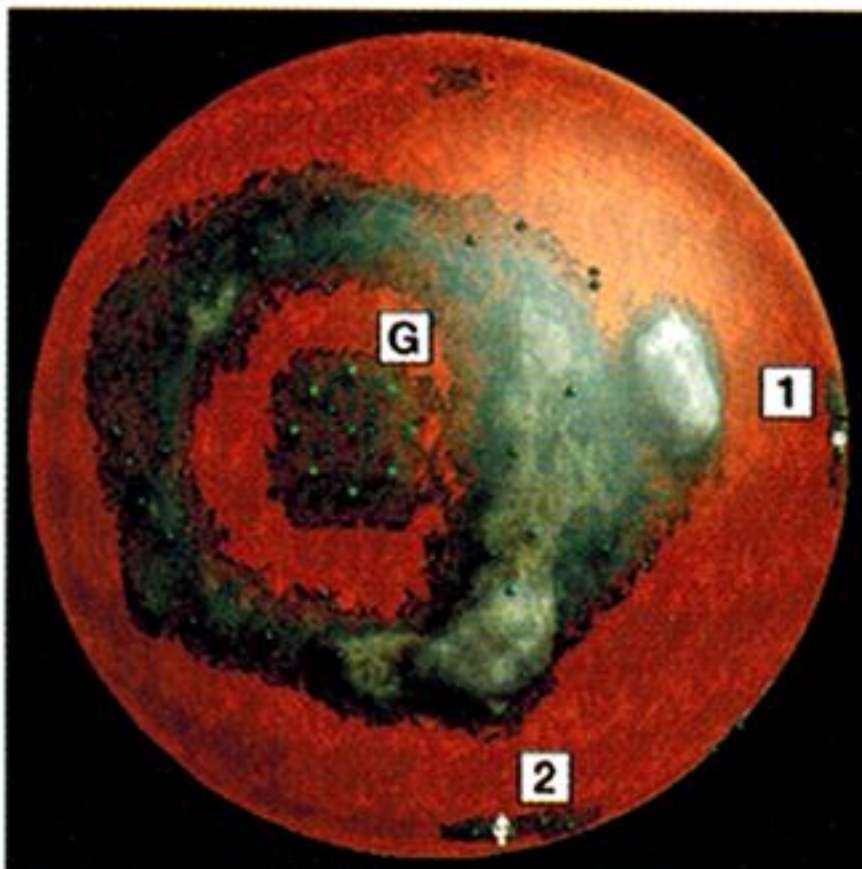
「ファイアストーム」スペルや「浸食」スペル，「火山噴火」スペルはストーンヘッドの礼拝で手に入れることを考えよう。

敵はボートあるいは気球にのって侵攻してくるはずなので，このとき「マジックシールド」をかけた従者で攻め込まれるとやっかいなので，「催眠」スペルも常にストックしておきたい。

どうやって侵攻するか

互いの陣地の島は海によって隔てられているので，結局，ボートや気球といった乗り物に乗って敵地へ侵攻を仕掛けることになる。

敵地へひっそり侵入

このレベルの概略図
1.「浸食」スペル（×3）
2.「ファイアストーム」スペル（×3）
3.「浸食」スペル（×3）
4.「ファイアストーム」スペル（×3）
5.「火山噴火」（×1）
RBGY．各軍勢のスタート位置。4軍勢ともに各陣地はこのような形をしている）

したいならばボートが最適だろう。ボートは5人を輸送でき、しかもそのすべての従者に「透明」スペルをかけてしまえば、敵の防衛ラインの目を盗んで敵地へ上陸することができるのだ。なお、上陸後、攻撃活動に移った瞬間に「透明」スペルの効果は切れるので、もし、敵地で激しく暴れ回りたいならば、同時に「マジックシールド」もかけておく必要があるだろう。

敵の防衛ラインの目を盗むのであれば、透明従者を乗せた気球でもいいのだが、気球はその存在が目立ちすぎるので、隠密行動には向かない。

ストーンヘッドを礼拝する

このレベルにはストーンヘッドが多く、どれも使いようによっては強力な攻撃手段となるスペルを与えてくれるのでぜひとも礼拝したいところだ。

それぞれの陣地から比較的近い位置にある、「穴(クレーター)の開いていない」4つの島のストーンヘッドは「浸食」スペル、「ファイアストーム」スペルをそれぞれ3つずつ与えてくれる。

それら4つのストーンヘッドの具体的な位置とそこで得られるスペルの対応を以下に示しておこう。

・プレイヤー1(青)の場合
西側のストーンヘッド…「浸食」スペル
北側のストーンヘッド…「ファイアストーム」スペル
・プレイヤー2(赤)の場合
東側のストーンヘッド…「浸食」スペル
北側のストーンヘッド…「ファイアストーム」スペル
・プレイヤー3(黄)の場合
西側のストーンヘッド…「浸食」スペル
南側のストーンヘッド…「ファイアストーム」スペル
・プレイヤー4(緑)の場合
東側のストーンヘッド…「浸食」スペル
南側のストーンヘッド…「ファイアストーム」スペル

これらのストーンヘッドの礼拝方法は2つ。

ひとつは乗りものに乗ってこの島へ行き、礼拝を行う方法だ。敵も全力でこれを妨害してくるはずなので、礼拝者の護衛部隊が必要になるだろう。

もっとも合理的で効果も高いのが、礼拝者をボートで島へ移動させて礼拝を開始させて、その上空を炎の戦士の気球軍団を停泊させる戦法だ。

この戦法を使えば、敵は一般従者でこの礼拝を妨害することはこれでまず不可能になるはずで、どうしても妨害したいという場合には敵シャーマン自身がこの地にやってくるしかなくなる。

もうひとつの戦法はこうだ。

まず、自軍陣地内の小高い丘に「土地隆起」スペルを駆使して登り、さらにこの丘からストーンヘッドのある小島へ「土地隆起」スペルを投じる。これによってストーンヘッドまでの陸橋ができるのでここを使って従者を移動させて礼拝させる。

一見すると遠くて「土地隆起」スペルは届きそうにないのだが、この丘に登ってしまえば、高地におけるスペル射程距離の伸長法則により、ぎりぎり届いてしまうのである。

さらに、このストーンヘッドのある島は、すでに述べたように比較的敵地に近いので、この島から「土地隆起」スペルを2発ほど使えば、敵陣本拠地と接続してしまうことができる。つまり、自軍陣地と敵陣地を陸続きにしてしまうことができるのだ。

この場合、こちらが敵地に直接侵攻を仕掛ける侵攻路を得たのと同時に、敵地から直接攻撃を受ける可能性も高まったという事実も把握しておくこと。よってこの作戦を実行する場合には、敵よりも圧倒的に兵力があることが大前提となる。

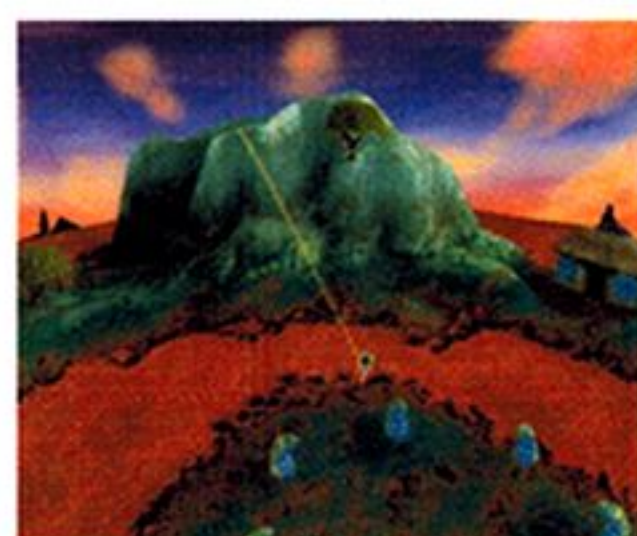
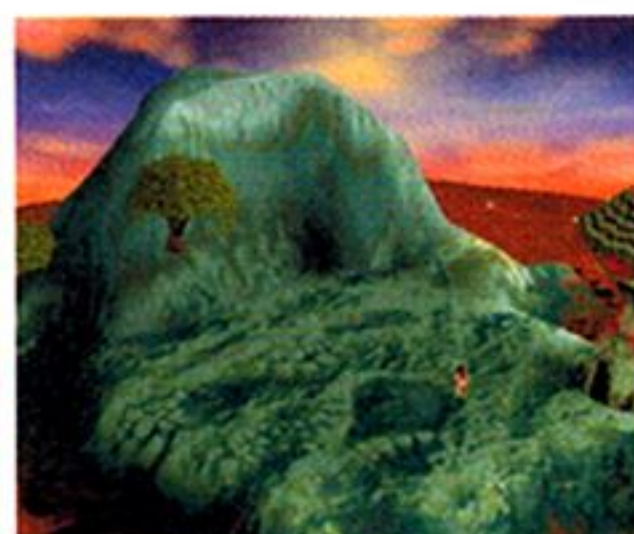
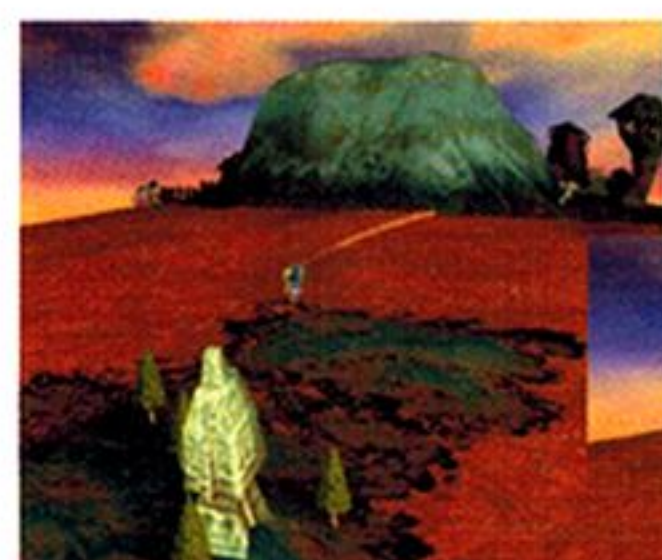
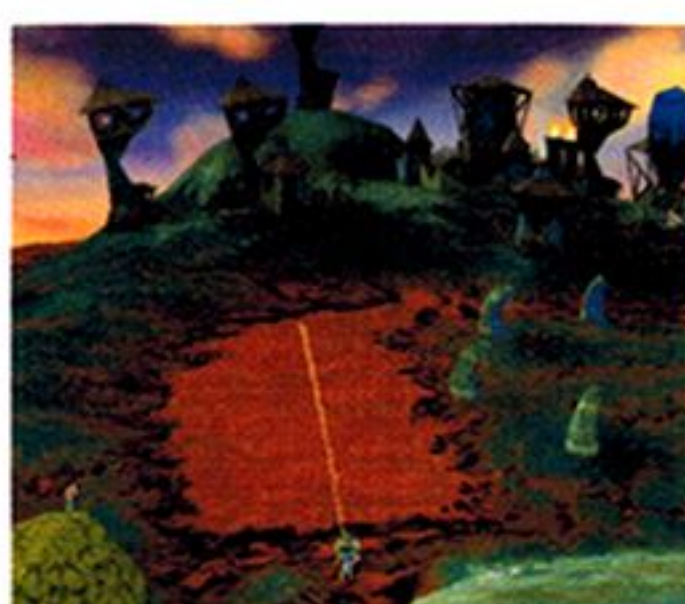
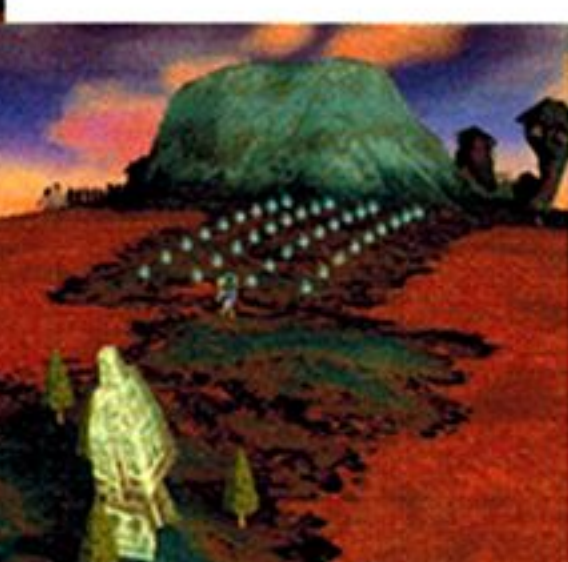
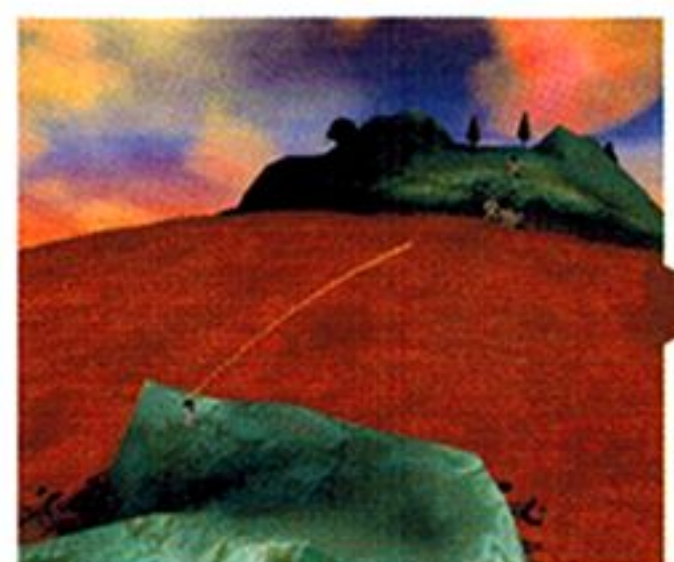
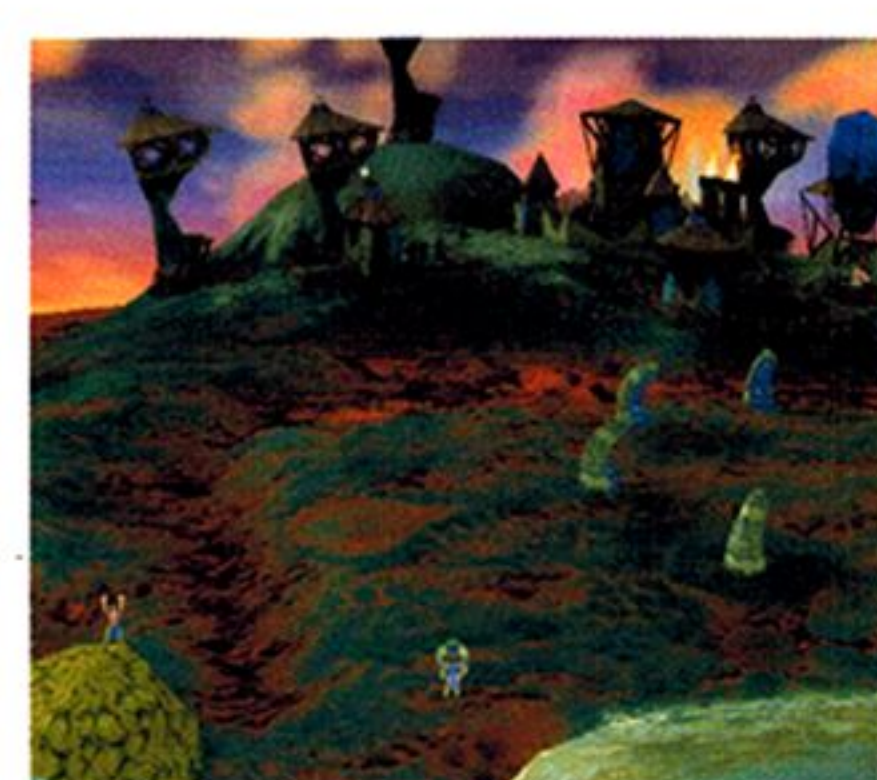
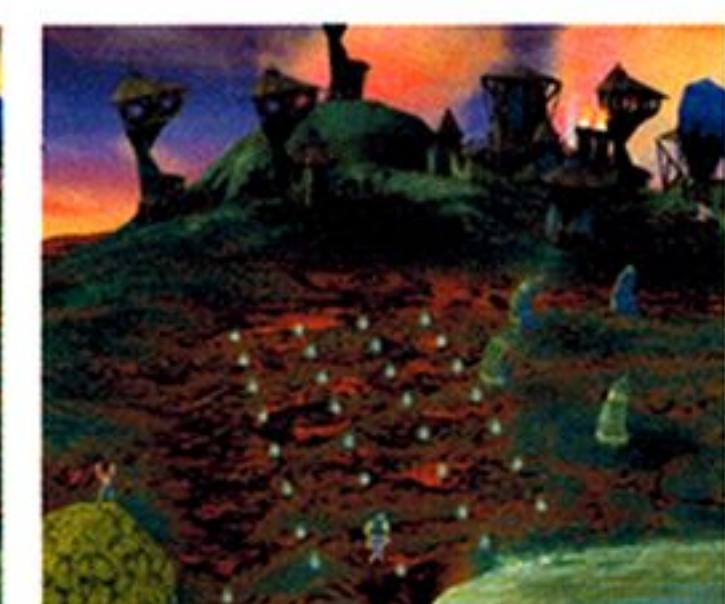
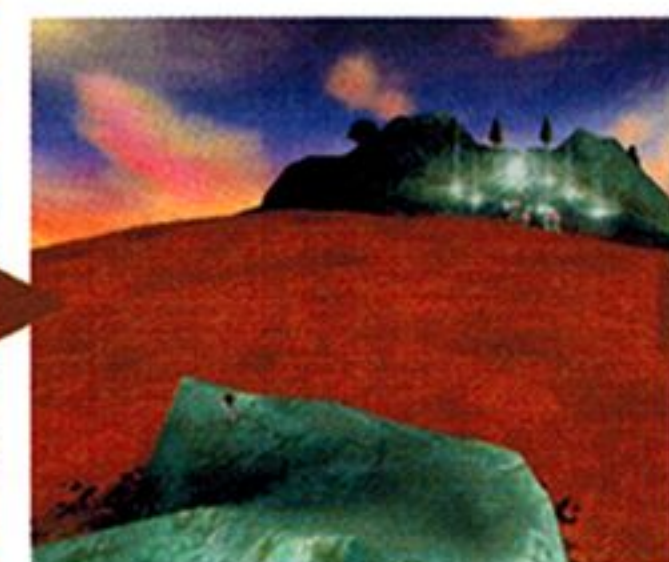
クレーターのある島のストーンヘッド

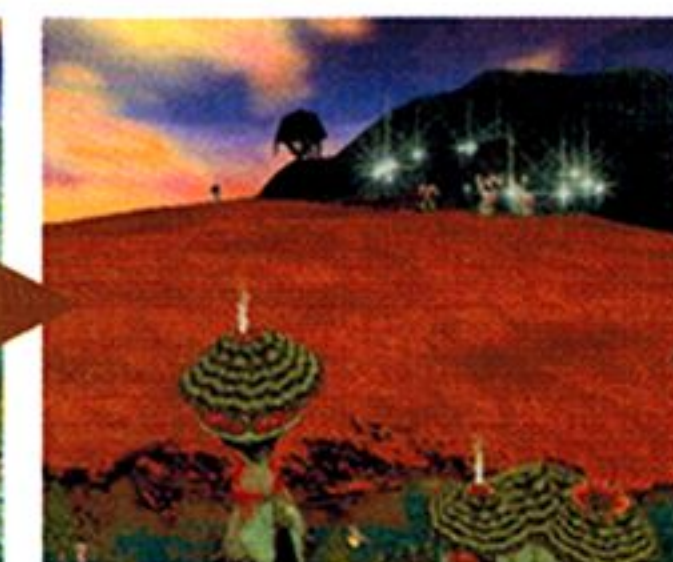
各軍勢の陣地の島に取り囲まれた格好で、やや大きめの島が存在している。この島には穴(クレーター)のあいた山のそびえたっており、そのふもとにはストーンヘッドがある。そしてこのストーンヘッドは強力無

ボートに乗せた従者に「透明」スペルと「マジックシールド」スペルをかけて敵地へ送り込む

敵のプレイヤーから見ると、無人のボートがこっちにやってくる、という異様な光景を目のあたりにすることになり、バレバレなのだが、防衛ラインはこれに気がつかない

「土地平坦」スペルで陣地内の丘を平坦にして敷地面積を稼ぐのもいいだろう

専有面積の広い建物は先に建ててしまおう

ストーンヘッドの島を経由して敵地と接続する

自軍陣地と敵陣地はストーンヘッドのある島を挟んで陸続きとなった

「土地隆起」スペルで自軍陣地内の池を埋めよう。これを繰り返せば建築スペースがちょっと稼げる

丘の上から島へ「コンバート」スペルを放ち……

勇士を獲得

これを「火山噴火」スペルを与えてくれるストーンヘッドに礼拝させる

シャーマンをガードタワーに登らせるだけで、この島へ「コンバート」スペルを届かせることができる軍勢もある

比な攻撃スペル「火山噴火」スペルを与えてくれるのだ。
何度もいっているように，このレベルの各軍勢の陣地は非常に狭いため，陣地中央に「火山噴火」スペルを発動されてしまうと確実に壊滅する。ほかのストーンヘッド以上にこのストーンヘッドは獲得したいところだ。

ゲーム開始早々に「火山噴火」スペルを手に入れる方法

さて，ここへの礼拝はもちろん，その他の4つのストーンヘッドと同様の戦法で礼拝すればいいのだが，このストーンヘッドだけに通用するちょっとトリッキーな礼拝方法があるので紹介しておこう。
まず，自軍陣地内の小高い丘に「土地隆起」スペルを放って斜面を作り出し，この丘に登る。陣地内にはいくつかの丘があると思うが，ここで登るべきなのは「火山噴火」スペルを与えてくれるストーンヘッドがある島に近い方の丘だ。
この丘に登ったら，その島に住む野人に「コンバート」スペルを放ち，自軍勇士にしてしまうのだ。そして，この勇士をそのままその島のストーンヘッドに礼拝させてしまうのだ。
「コンバート」スペルはもともとすごく射程距離が長いので，丘に登るとさらに射程距離が伸び，なんとこの島にまで届いてしまうのだ。
礼拝は開始されるものの，気球軍団の護衛部隊を送らないといずれは敵に礼拝を邪魔をされてしまうだろう。しかし，それでも，「ボートを作って礼拝者を乗せて，島へ移動させて……」という手間は省くことはできることになる。
なお，このストーンヘッドの礼拝を成功させると，「火山噴火」スペルがストックされると同時にこの島の山が噴火を起こし，礼拝者は死亡してしまう。いわば生け贄というやつだ。シャーマンをここに礼拝させるのはやめておいたほうがいいだろう。

ストーンヘッドの島と自軍陣地を陸続きにしてしまう

礼拝が成功すると「火山噴火」スペルが手にはいるが……

突然ストーンヘッドの後ろで火山が爆発。礼拝者は全員そのマグマをかぶって死んでしまう

死海
DEAD SEA

心がまえ

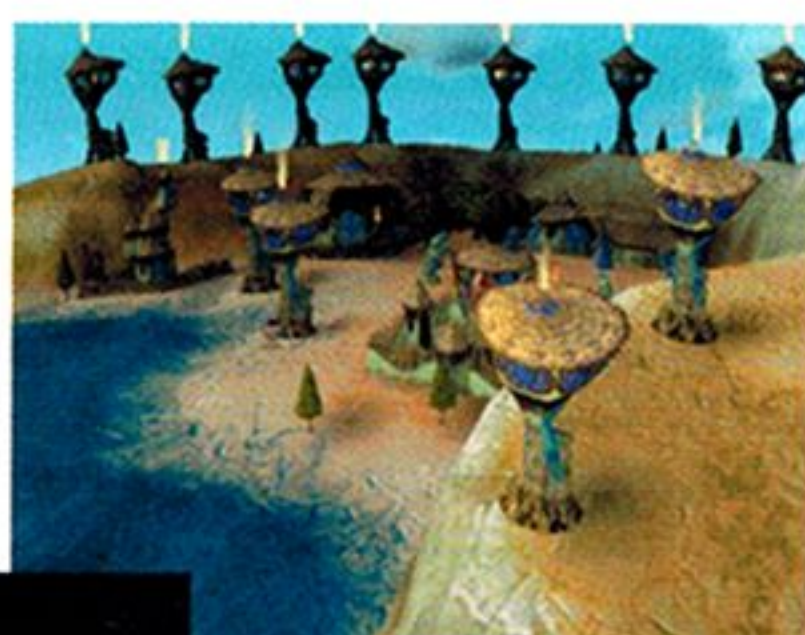
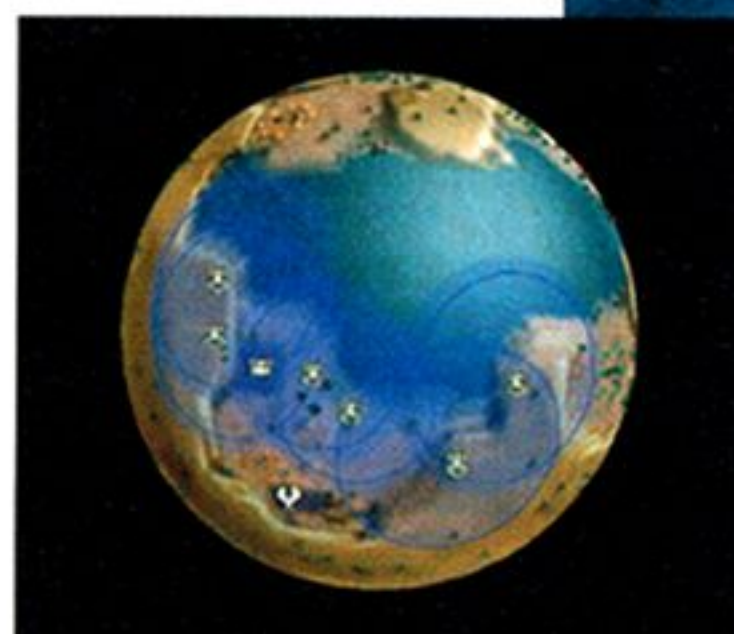
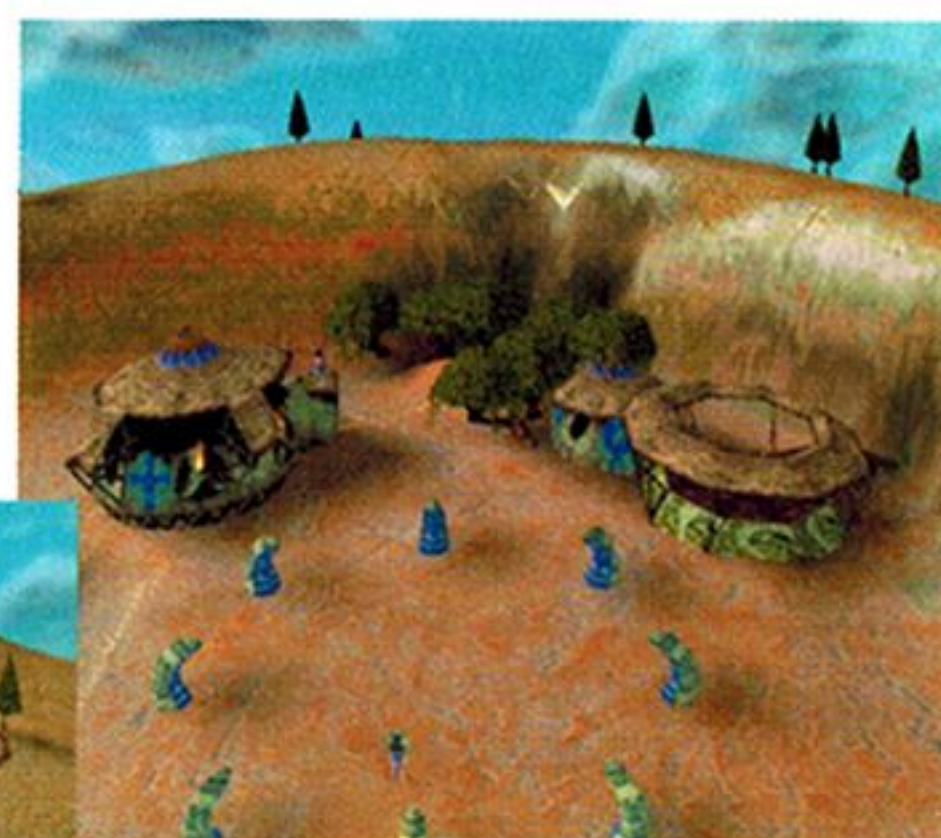
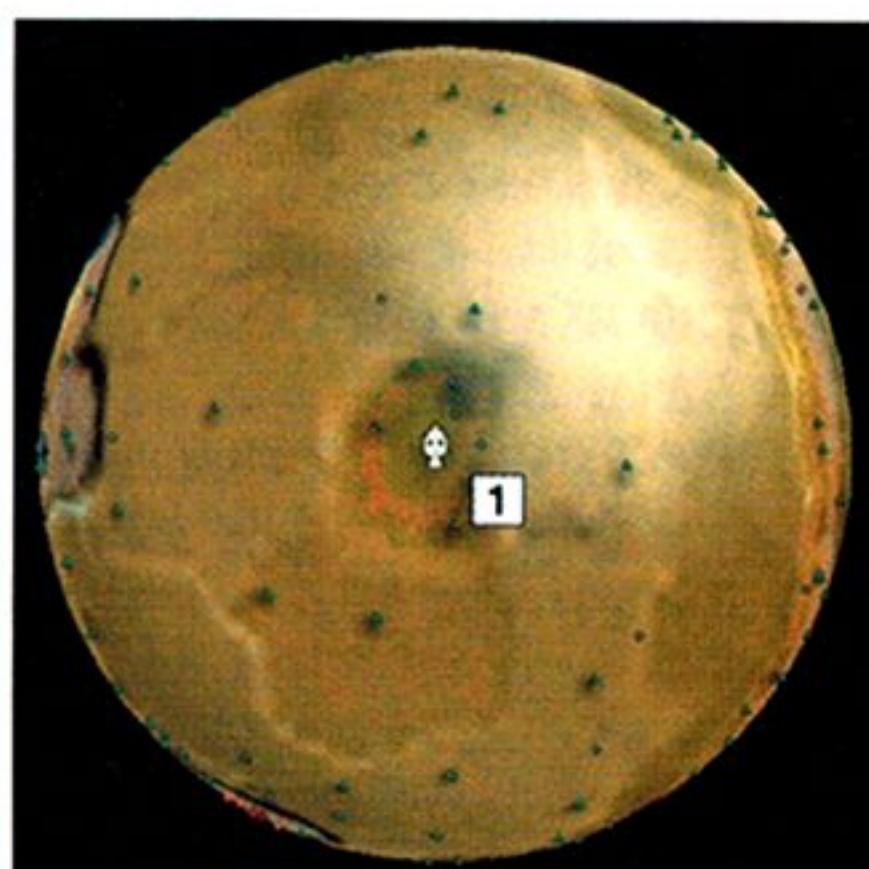
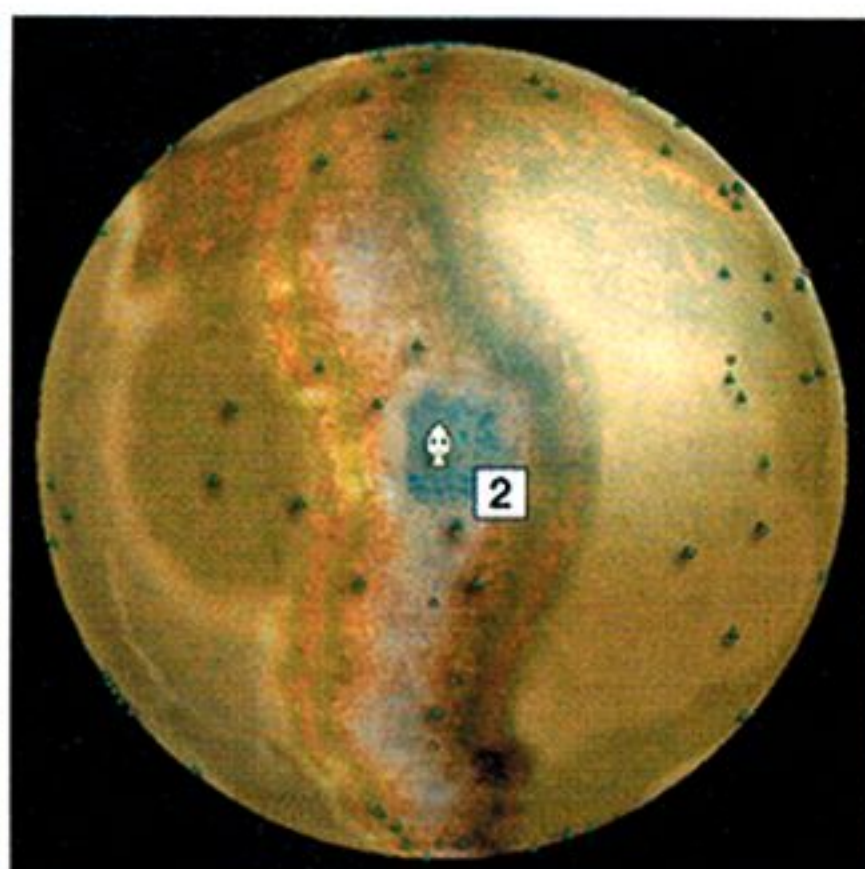
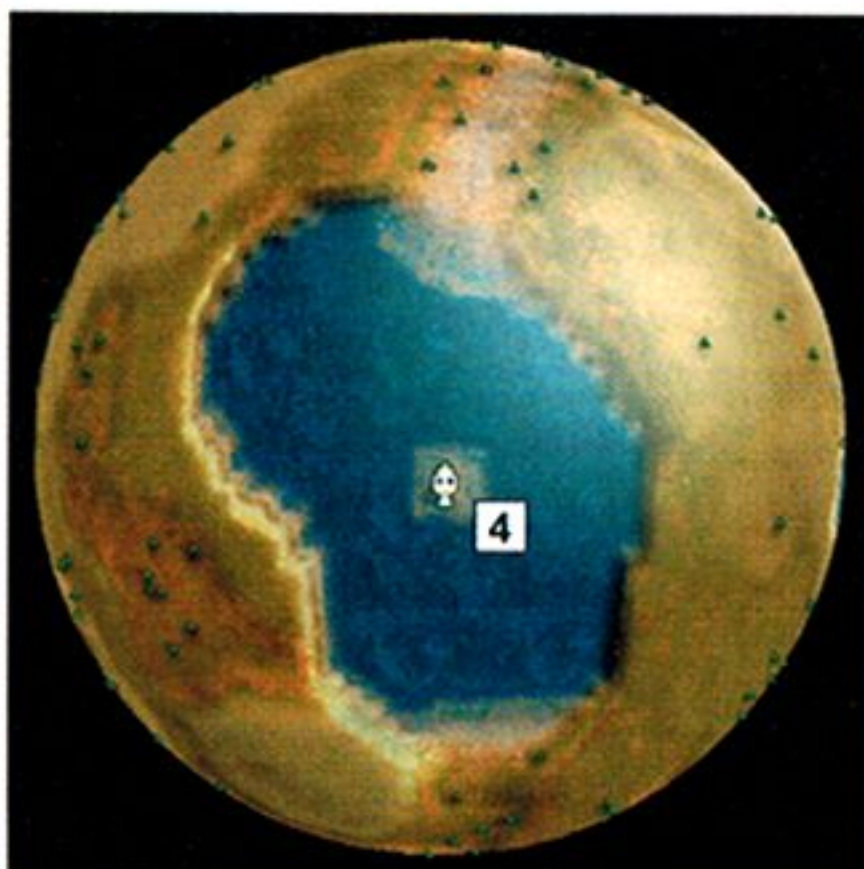
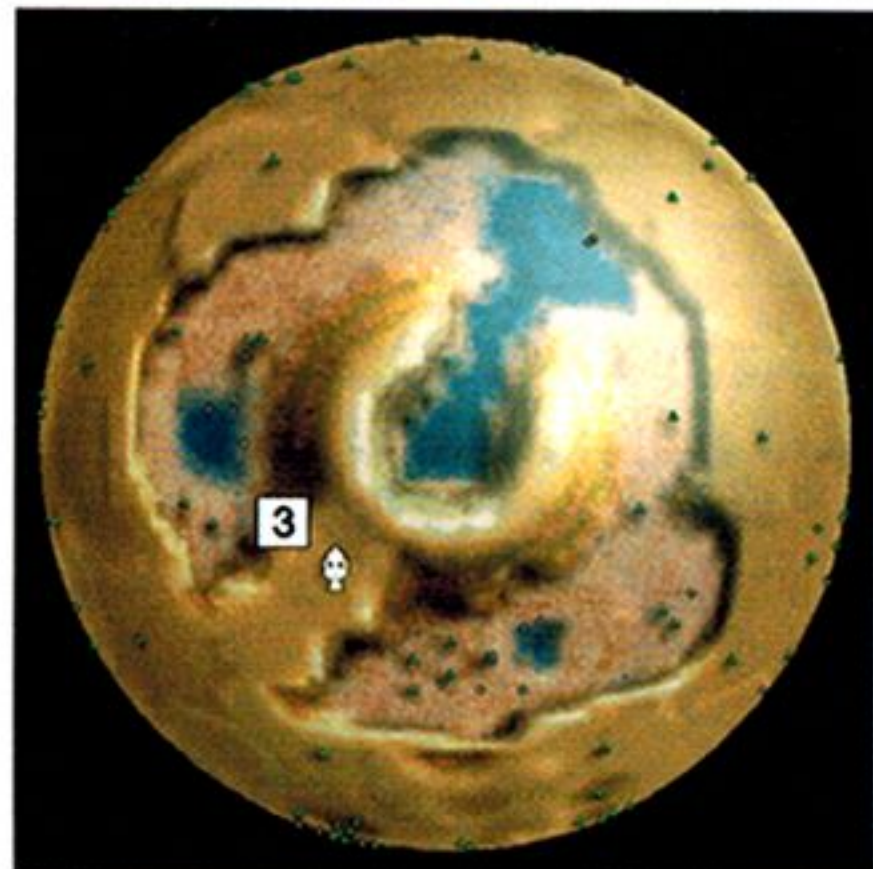
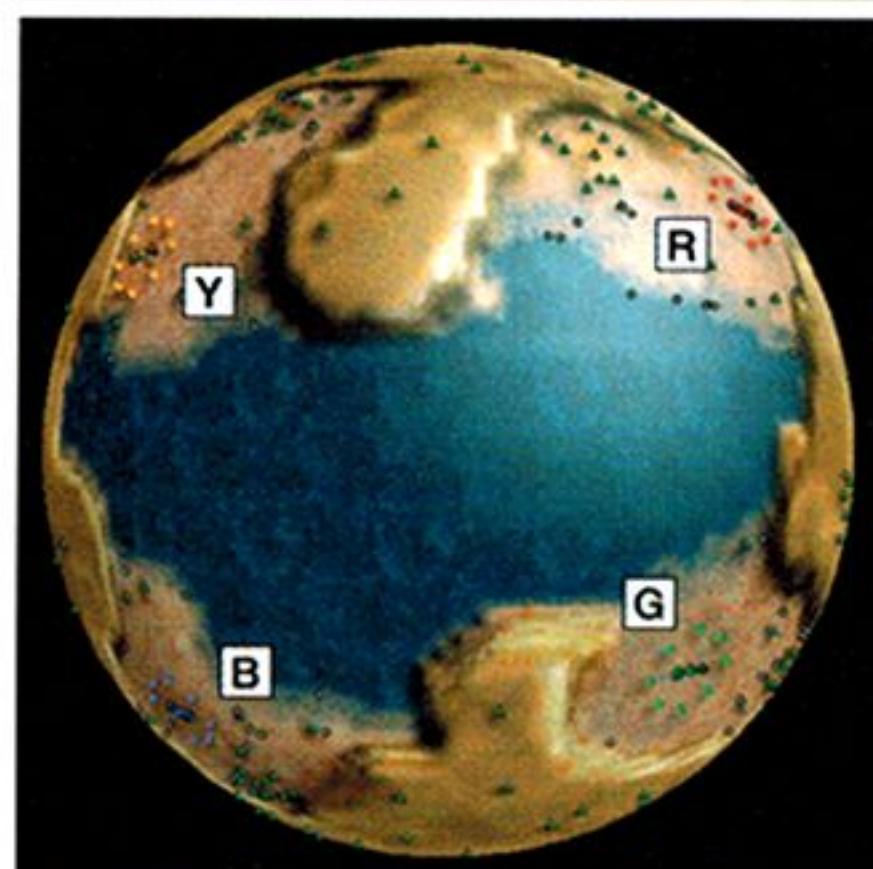
全世界のほとんどが地面という広大なレベル。ストーンヘッドは4つあり，そのすべてが強力な攻撃スペルを与えてくれる。ぜひとも入手すること。とくに「血の欲望」スペルは手に入れたい。「死の天使」スペルと「火山噴火」スペルは取得回数が有限なので早い者勝ちということになる。
互いの陣地は陸続きになっているので早期から侵攻が可能だ。速攻も十分有効だろう。
ゲーム開始時に与えられる各陣地は，谷底で非常に狭いので勢力はその外の広大な台地に広げる必要がある。

建設戦略

全軍は陣地として狭い谷底の地を与えられる。木々は豊富だが，敷地面積は小さいのでこの地に建てる建物は必要最低限のものにしておく必要がある。
このレベルはすべての敵陣と陸続きなので，伝道師

最初から使えるスペル	
火山噴火	ライトニング
死の天使	土地隆起
ファイアストーム	マジックシールド
浸食	透明
地震	ハチの大群
土地平坦	幽霊軍隊
腐食	コンバート
トルネード	ブラスト
催眠	

最初から建てられる建物	
ガードタワー	スパイ訓練小屋
小屋	気球小屋
伝道師訓練小屋	ボート小屋
戦士訓練小屋	ガードポスト
炎の戦士訓練小屋	

このレベルの概略図
1.「地震」スペル（回数無制限）
2.「死の天使」スペル（×5）
3.「血の欲望」スペル（回数無制限）
4.「火山噴火」スペル（×1）

尾根の上の防衛ラインは本拠地の低地を守るほか，その周囲の台地も守ってくれる

崖の上に建てたガードタワーに炎の戦士を住まわせるとものすごい広い範囲を防衛してくれる

低地から台地への出入り口となる斜面には伝道師，スパイなどを住まわせたガードタワーを配置し，敵の侵入を防ぐ

の速攻が考えられるのでまずは伝道師小屋を作っておくべきだ。

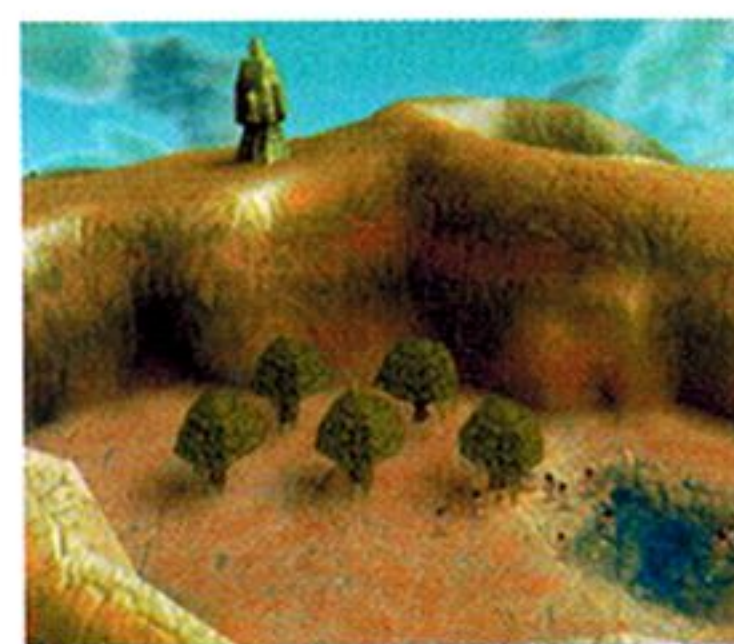
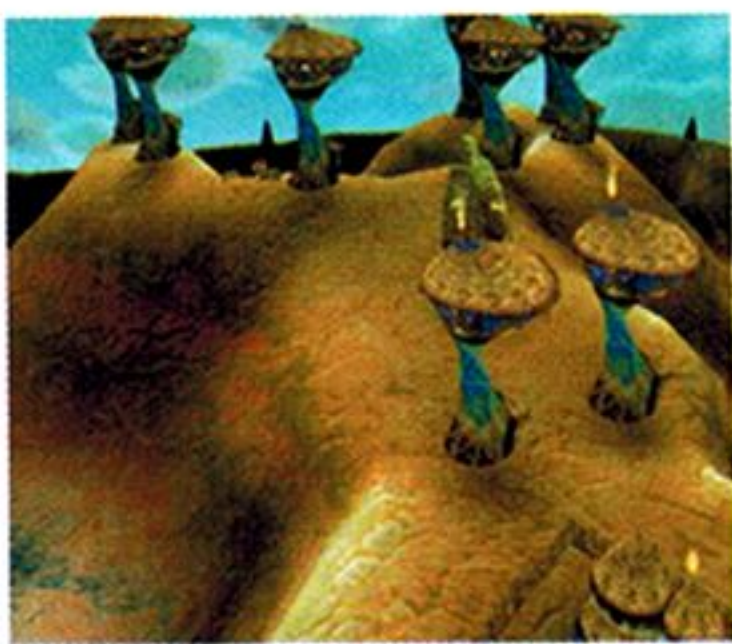
それぞれの陣地の谷底には広葉樹が密集している地帯があると思う。これは乗りものを作る際の木材採取場としてとても便利なので、この近くに気球小屋を作るといい。気球がスピーディに量産できるようになるはずだ。

なお、ボート小屋はこのレベルでは不要だ。敵陣とは陸続きなので、わざわざ攻撃目標になりやすいボートに乗って移動する理由もないからだ。

低地内の建設が軌道に乗り始めたら、今度はその谷底から出て、広大な台地の上でテリトリーを広げていくこととなる。

直接行って礼拝しているとただの攻撃目標になってしまう

このように街を作り防衛ラインを形成してから祈れば敵もそう簡単には手は出せない

本拠地の尾根外周に建てた防衛ラインの近くに小屋や各種訓練小屋を建てるといい。この防衛ラインは本拠地のある低地のほか，この尾根の周囲も守備範囲となっているからだ

ここが「血の欲望」スペルを無限に与え続けてくれるストーンヘッドのある場所

これだけの防衛ラインを構築してしまえば，このストーンヘッドは間違いなく自軍のものにできるだろう

木々が豊富なので，ストーンヘッドのある山の麓の低地を植民地として開拓しよう。ストーンヘッドの近くに築いたガードタワーの炎の戦士が麓の集落も守ってくれる

防衛ライン

自軍本拠地の低地は狭いが、ゲーム開始時に各種訓練小屋や気球小屋などを建設していると思うので、ここを叩かれると少々痛い。よってここを守る防衛ラインを築いておこう。

この本拠地の低地を守る防衛ラインは、この本拠地のある低地の上の外周、尾根の部分にガードタワーを建て並べていくことで形成する。ガードタワーには炎の戦士を中心に住まわせておくといい。

そして、この谷底に出入りするための斜面は限られているので、この斜面の上のガードタワーには伝道師とスパイのコンビを住まわせておくと、陸路から侵攻してくる敵の足止めができる。

対岸の敵陣の進攻に対する防衛ラインは、海側の崖の上に炎の戦士を住まわせたガードタワーを設置してあれば、それほど念入りに構築しなくてもいいだろう。この崖の上のガードタワーは非常に高度が高くなるため、ここに登った炎の戦士達の防衛範囲は海のほうにまで及んでいるからだ。

とはいうものの、海岸の中央付近はやや防備が薄くなるので、ここに2、3本のガードタワーを建て炎の戦士を住まわせておくといいだろう。これで陣地の防衛は完璧だ。

ストーンヘッドを植民地の出発点とする

本拠地の防衛ラインを完璧にしても、敵シャーマンの襲来や、「透明」スペルをまとった敵従者に侵攻される可能性も十分にある。「ここはいつか破壊されるかもしれない」と認識し、自軍のテリトリーをこの本拠地周辺だけではなく広大な台地に向かって広げていこう。

それでは、いったいこの台地のどこをどう開拓していけばいいのだろうか。

いちばん単純なのは、谷底のすぐ外から小屋などを建てていき勢力を広げていく方法だが、本拠地に近いと、万が一、本拠地が攻撃を受けた場合、そのまますぐここを攻撃目標にされて、同時に壊滅させられる危険がある。

そこで、自軍本拠地とは離れた場所に第2のテリトリーを作るのだ。さて、その具体的な場所だが、おすすめなのが、この世界に4つあるストーンヘッドの周辺だ。

まず、ターゲットとなるストーンヘッドの周辺にガードタワーを建て、この周囲にどんどん小屋や訓練小屋を建てていくのだ。敵の侵攻に備えて適当な間隔でガードタワーを建ててそこに炎の戦士や伝道師、スパイなどを住まわせておくといいのはいうまでもない。

そしてタイミングを見計らってそのストーンヘッドを礼拝しよう。

礼拝物を礼拝すると全敵プレイヤーにその事実が情報アイコンで知らされてしまうが、この戦法ならば、敵が気づいたときには「時すでに遅し」、となる。

そのストーンヘッド周辺にはすでに街が広がっているので、直接この礼拝を邪魔するのは難しくなってしまっているからだ。

自軍はこれで当分は安全に礼拝できることになり、強力なスペルが入手できることになる。

おすすめストーンヘッドはどれ?

やはり「血の欲望」スペルを与えてくれる、概略図でいうところの3番のストーンヘッドのポイントがいちばんの植民地候補地だといえる。

まず、その理由として「血の欲望」スペルを回数無制限に与えてくれるという点が挙げられる。

「血の欲望」スペルをかけられた従者は攻撃力と防御力が3倍に高められ、移動能力も倍増、さらに敵伝道師に洗脳されないという特典も獲得する。これを戦士にかけて敵陣に送り込めば、敵地に壊滅的な被害をもたらすことができるわけで、とにかくこのストーンヘッドを敵に受け渡す理由はない。

また、ここを候補地に挙げる第2、第3の理由として、木々が比較的豊富だという点、野人が多く生息している点が挙げられる。この周囲にテリトリーを広げるには建物を建てる必要があるわけでそれには木材と人員は欠かせない。木々と人員が豊富ならばそれだけ早くテリトリーを拡大できるわけで、それはつまり、早く「血の欲望」スペルを手に入れられることにも繋がっていく。

さて、この地における具体的なテリトリー拡大方法だが、まず、ガードタワーを1本、このストーンヘッドの正面に建てることから始めよう。そしてこのガードタワーを基盤としてどんどん小屋を建て並べていく。同時に、ストーンヘッド背後のU字型の尾根にもガードタワーを建て並べ、その麓の低地にも建物を建てていくといい。

それぞれのガードタワーには炎の戦士を住まわせておけば、これは自動的に礼拝者を守る護衛部隊にもなり、また、麓の街を守る防衛ラインとしても機能する。

フェイスオフ
FACE OFF

心がまえ

各陣地は陸続きで、その中央には「火山噴火」スペルを与えてくれるストーンヘッドがある(以下ストーンヘッド5)。このレベルで「火山噴火」を使われるとまず壊滅状態に追い込まれるので、決して敵にこれを取られてはならない。

各陣地の近くには、スパイ訓練小屋やボート小屋の設計図を与えてくれるオベリスクが建っている小島があり、ここへは「土地隆起」スペルで行くことができる。

各陣地は狭いが、世界全体も狭い。このレベルは短期決戦用ということができるだろう。

野人の確保

ゲームが開始されたら自軍陣地内の野人はそのままにして、ストーンヘッド5のある中央の山に向かおう。ここには野人が密集しており、絶好の野人獲得場所となっている。このレベルではこの山の頂上以外に、自由な土地がないので、敵に序盤で人口の差をつけるには、ここの野人をコンバートする以外に方法がないのだ。

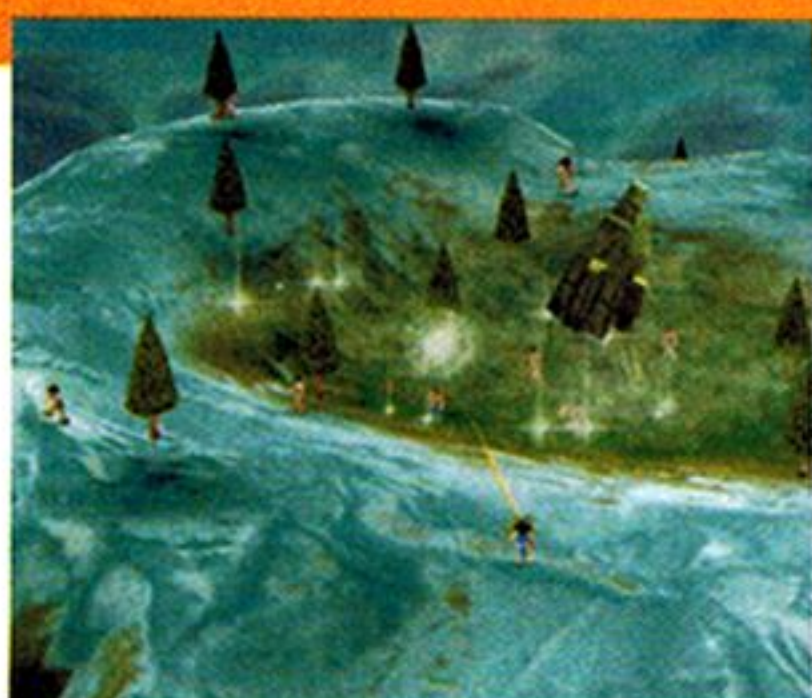
防衛ラインの構築

ゲーム序盤は、敵が侵攻をしかけてくる経路はたったひとつ。ストーンヘッド5のある山のほうからだ。

最初から使えるスペル	
ファイアストーム	土地隆起
土地平坦	透明
腐食	幽霊軍隊
催眠	コンバート
ライトニング	ブラスト

最初から建てられる建物	
ガードタワー	炎の戦士訓練小屋
小屋	スパイ訓練小屋
伝道師訓練小屋	ボート小屋
戦士訓練小屋	ガードポスト

防衛ラインは、この進入路となる斜面に構築する。ここにガードタワーを2列ほど建て、ストーンヘッド5に面した前列側には炎の戦士を住まわせ、後列には伝道師、スパイ訓練小屋入手後はスパイを住まわせるといい。

また、敵がこのストーンヘッド5に無関心ならばこ

絶好の「野人狩りの場」となるストーンヘッド5のある山の頂上

「火山噴火」スペルと引き替えに，その周辺の防衛ラインを失うことになる

のストーンヘッドを取り囲むようにしてガードタワーを建てて，そこに炎の戦士を住まわせると面白い。これによりストーンヘッド5の礼拝を優位に行うことができ，仮に礼拝が成功しなくても敵プレイヤーの意識をそこに集中させることができる。

また，このストーンヘッド5のある山から敵地へと続く斜面の上にガードタワーを建ててしまい，そこに炎の戦士や伝道師を住まわせるという「イヤガラセ行為」をやるのも面白い。これが完成すると敵従者が自分達の陣地から出ようと斜面を登ったときから早々と攻撃を受けることになり，敵はこれをどうにかしない限り自分たちの陣地へ閉じ込められた格好になるのだ。これはいわば一種の「攻撃的な防衛ライン」といえるかもしれない。

「火山噴火」スペル

ストーンヘッド5は礼拝することで「火山噴火」スペルを与えてくれるが，ひとつ問題点がある。

「火山噴火」スペルを与えてくれたあと，なんとその場から噴火が始まってしまうのだ。もちろん獲得した「火山噴火」スペルは自分の自由に使えるのだが，それとは別に強制的にストーンヘッドのあった場所から噴火が始まってしまうのだ。

流れ出たマグマは，この山の上に築いた防衛ラインを破壊してしまうので，これを礼拝するのは一長一短だといえる。

もし，「自分でも取りたくないが敵に『火山噴火』スペルを取られるのはちょっと……」というのであれば，このストーンヘッドの前面に「腐食」スペルを大量に仕掛けておき，敵の礼拝を阻止するといいだろう。

オベリスク

このレベルには各陣地の近くにオベリスクが建っている離れ小島がある。

それぞれなにを与えてくれるかは以下のとおり。

・プレイヤー1（青）の場合
　北側のオベリスク…「スパイ訓練小屋」の設計図
　南側のオベリスク…「ボート小屋」の設計図
・プレイヤー2（赤）の場合
　東側のオベリスク…「スパイ訓練小屋」の設計図
　西側のオベリスク…「ボート小屋」の設計図
・プレイヤー3（黄）の場合
　南側のオベリスク…「スパイ訓練小屋」の設計図
　北側のオベリスク…「ボート小屋」の設計図
・プレイヤー4（緑）の場合
　西側のオベリスク…「スパイ訓練小屋」の設計図
　東側のオベリスク…「ボート小屋」の設計図

このオベリスクの立っている離れ小島には「土地隆起」スペルで陸橋を作っていくことになる。

スパイ訓練小屋の活用方法

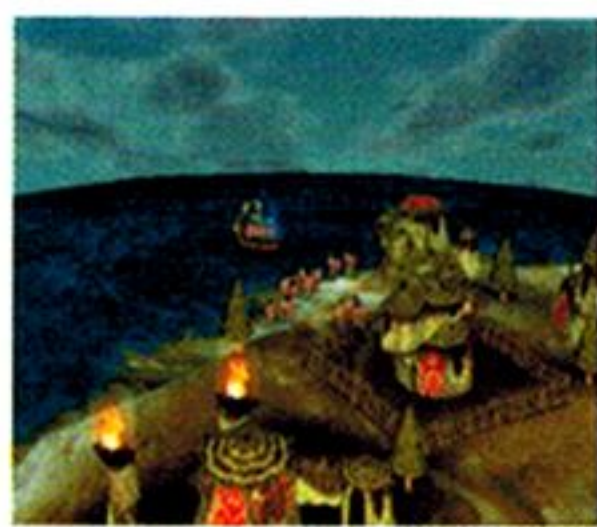
オベリスクを礼拝することで入手した「スパイ訓練小屋」と「ボート小屋」をどう活用したらいいのかを考えてみよう。

スパイ訓練小屋は敵従者になりすまして敵施設の放火活動を行うことができる従者「スパイ」を育成するところだ。使いようによってはかなり強力な戦力となるスパイだが，このレベルでは活動がしづらい。

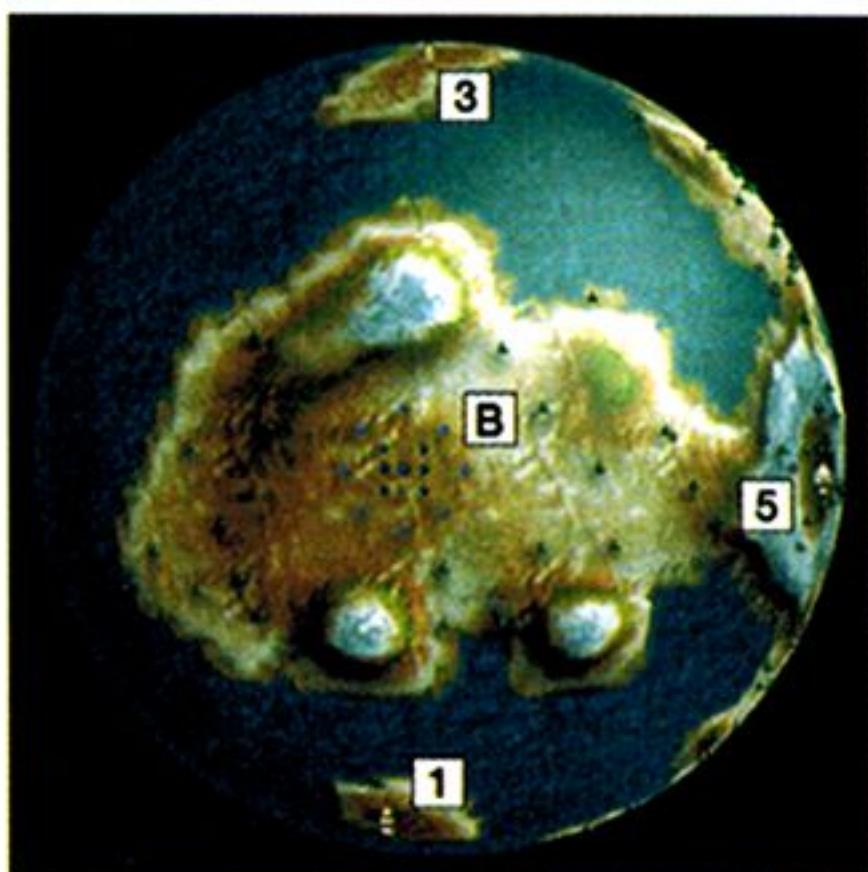
スパイは敵従者に変装してから放火活動を行うが，その瞬間を敵従者やプレイヤーにみつかると瞬く間に殺されてしまう。土地が狭く各建物が密集して建設されがちなこのレベルでは，敵プレイヤーも敵の従者も陣地内の異変には迅速に対応してくるため，すぐに見つかってしまうのだ。

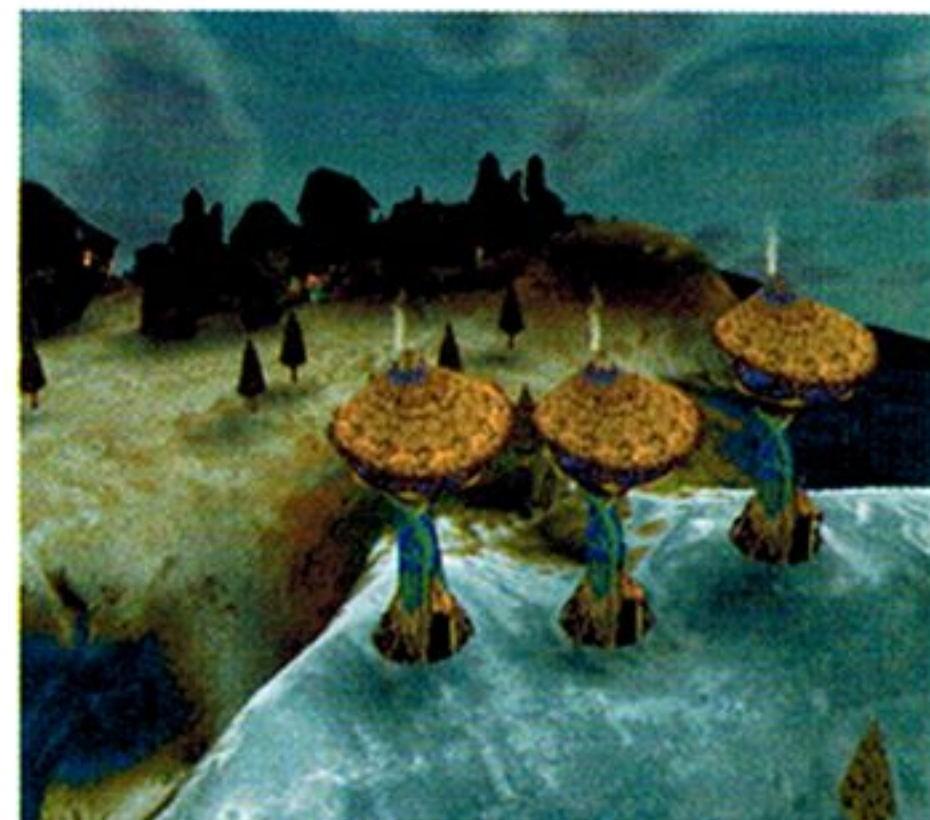
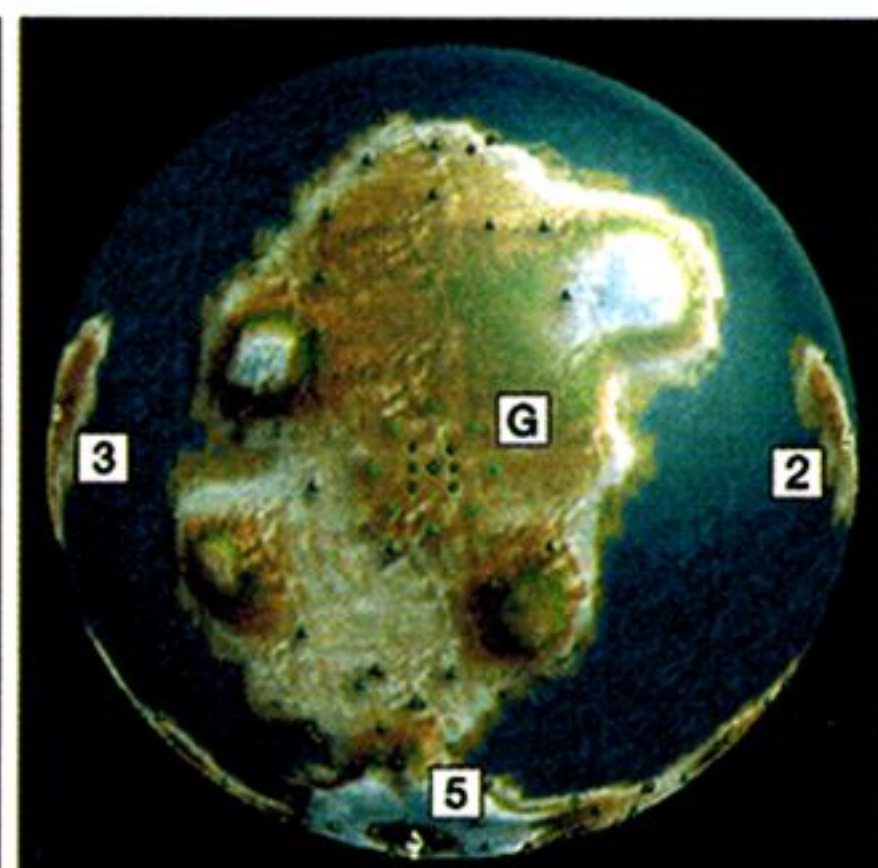
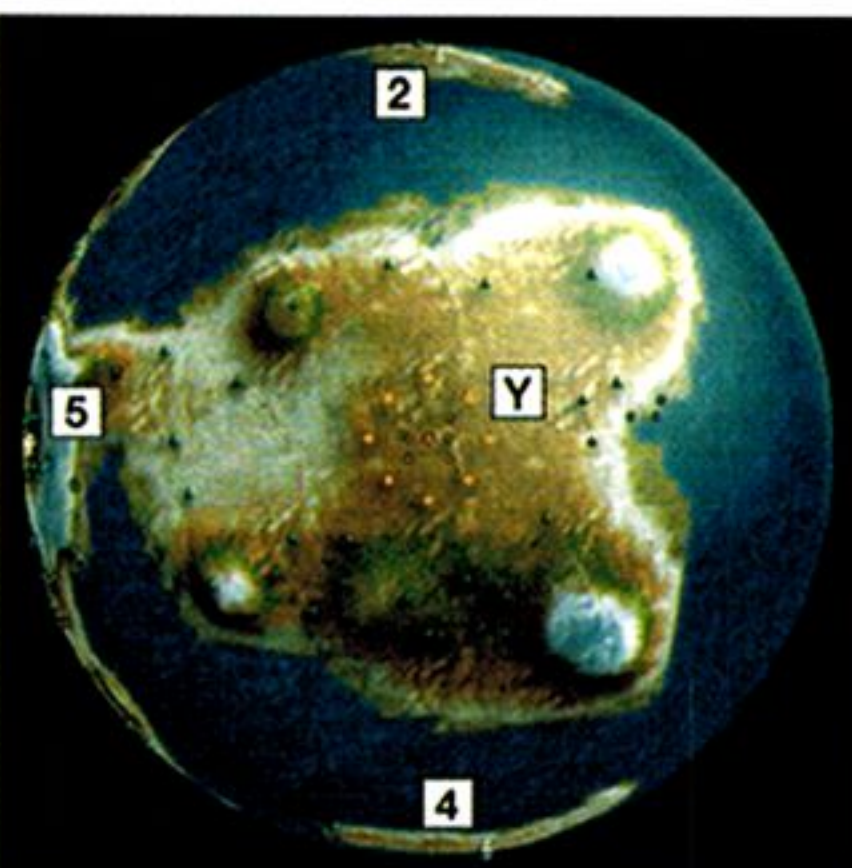
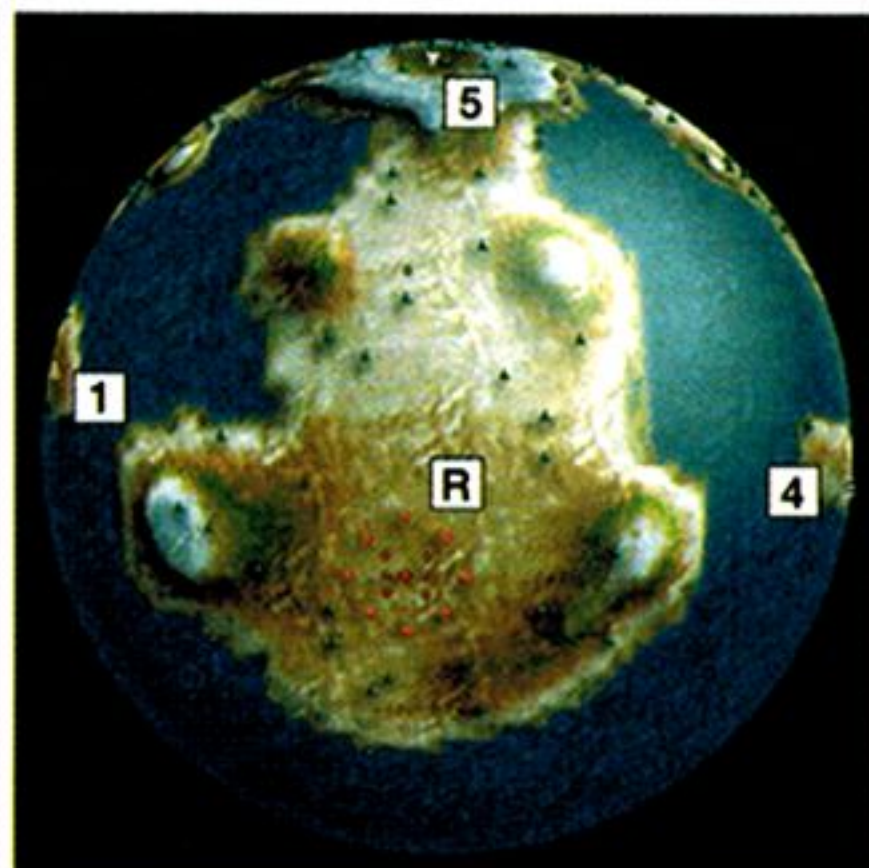
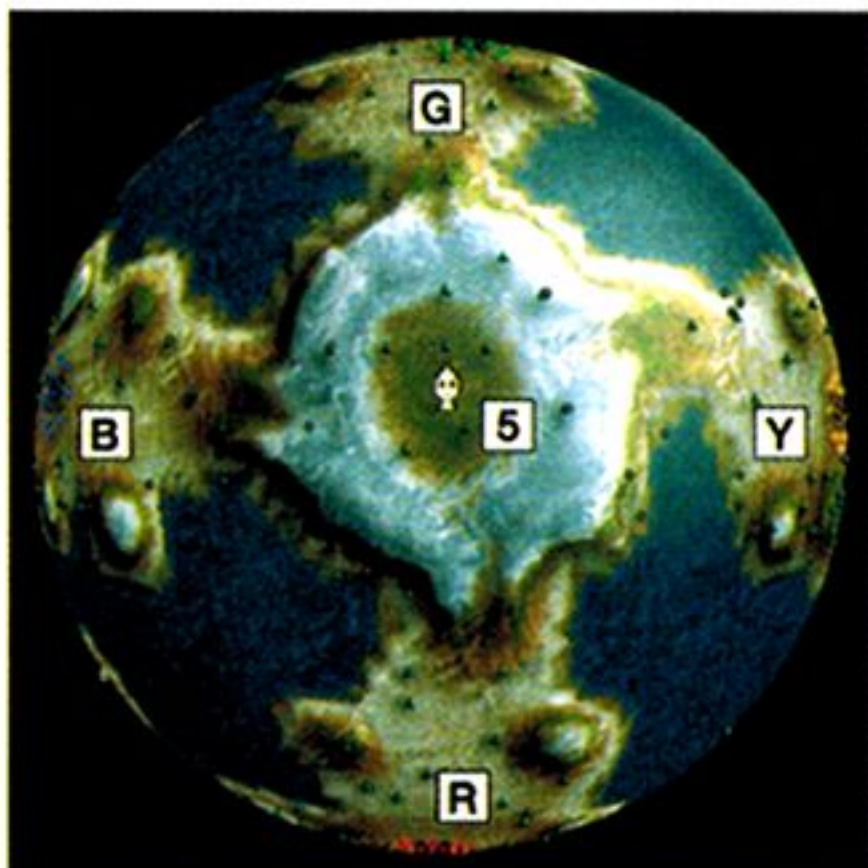
とはいえ，高確率で1回目の放火は成功することが多いので，ときどき敵伝道師訓練小屋などの重要施設の放火に挑戦してみるといい。

とにかく，敵がスパイ訓練小屋をまだ手に入れていない状況であれば，敵はスパイを持っていないわけで，見破る手段は限られてくる。戦いを優位に進めていけるはずだ。

ボート小屋の活用方法

「陸続きのレベルなので，ボート小屋なんて使わないのでは？」と思うかもしれないが，それこそが狙い目なのだ。

ゲーム開始状態では誰も乗りものを持って

このレベルの概略図
1.「ボート小屋」設計図（×1）
2.「ボート小屋」設計図（×1）
3.「スパイ訓練小屋」設計図（×1）
4.「スパイ訓練小屋」設計図（×1）
5.「火山噴火」スペル（×1。ただし，ストーンヘッドのある場所から噴火が起こり周囲に被害をもたらす）

このレベルの防衛ラインの基本形

ストーンヘッド5をこちらのガードタワーで取り囲んでしまえば礼拝が行いやすくなる

敵陣からストーンヘッド5の山の頂上へ続く道をガードタワーでふさいでしまう。攻撃的な防衛ラインの完成だ

いないため、全プレイヤーが「敵はストーンヘッド5のある山の斜面から侵攻してくる」と思い込んでいる。そのため、ほとんどのプレイヤーが防衛ラインを、この斜面の上にしか築いていないはずだ。

ボート小屋を手に入れたならば、ボートを作り、ここに伝道師を乗せて、これを防衛ラインの構築されていない敵陣地の海側から上陸させてしまうのだ。あるいはシャーマン自身がこのボートに乗って敵地に上陸し攻撃スペルで暴れまくるというのも有効だろう。

また、同時多角攻撃で、その斜面の方面からも敵地に同時に侵攻させるのも面白い作戦だ。きっと敵プレイヤーは目の回るような忙しい局面を迎えることだろう。

裏侵攻ルート

裏から侵攻を仕掛ける方法は、なにもボートを使った戦法だけではない。

オベリスクの立っていた離れ小島に「土地隆起」スペルを使って陸橋を作りここを渡っていったわけだが、その離れ小島の反対岸に行くとあることに気づくはずだ。

そう、海の向こうではあるが、敵地がきわめて近い位置に見えるのだ。実はこの反対岸からこの敵地へは「土地隆起」スペルが届いてしまうのである。

つまり、自軍陣地と敵陣地の間に、ストーンヘッド5のある山を経由しない侵攻路を作り出せてしまうことになるのだ。

もちろん、敵がこれを予測して、ここのに防衛ラインを構築している場合もあるので注意すること。

構築していない場合は、ここから、こちらの兵力を一気に送り込み敵地を壊滅させよう。この侵攻路は敵に使われる可能性もあるので、侵攻路を作った側は、その最初の襲撃で勝負をつけられるほどの兵力を持っていなければならないだろう。

圧点
PRESSURE POINT

心がまえ

4つの軍勢が正方形上の頂点の位置に存在し、敵地とは直線状の陸路で直結されている。シングルプレイヤーゲーム用のレベル17によく似たレベルだ。

各陣地に取り囲まれた中央の丘にあるストーンヘッドは「ハルマゲドン」スペルを与えてくれるが、参加プレイヤーがよほどのお人好しでない限りは、ここの礼拝はシングルプレイヤーのときのようにうまくはいかない。ここに固執すると兵力を消耗するだけだ。

野人、木々ともに豊富なので、早く勢力を整えた軍勢が優勢になりやすい。マップ自体は単純明快な構造をしているので、ある意味、戦略の優劣よりもマウス裁きやキーボード操作の早さが試されるレベルともいえるだろう。

野人を大量に獲得せよ

このレベルは非常に野人が豊富だ。一度コンバートし尽くしたはずの土地でも、しばらくすると再び自然発生する。とにかくゲーム序盤から終盤にかけて、野人を見つけたら、たとえひとりでもどんどん勇士として確保してしまおう。

野人は、各陣地間を結ぶ陸路にたくさん群れている。自軍陣地内の野人コンバートはあと回しにして、とにかくこの陸路上の野人を先にコンバートすることに専念しよう。このレベルは木々が豊富なので、序盤の勇士の人数が多いほうが圧倒的に先に建設事業が進むのだ。

自軍陣地近辺の陸路の野人をコンバートし尽くしたら、貪欲に敵地近辺の野人もコンバートしていってしまおう。

また、ときおり敵シャーマンが自分たちの勇士に指示を出すのに夢中なのか、この陸路上で立ち止まっていることがある。このときは容赦なく「ブラスト」スペルで攻撃してしまおう。敵シャーマンは吹っ飛び、かなりの高確率で陸路両サイドにある池に沈めることができる。序盤に敵軍との勢力差をつけるためにも、敵シャーマンを見かけたらどんどん狙っていくといい。

防衛ラインの形成

木々は、各陣地を結ぶ陸路上に豊富なので、防衛ラインの中核となる建物であるガードタワーは、かなり多めに建てることができる。

そこで、防衛ラインは2段構成のものを作るといい。具体的にはこうだ。

まず、自軍陣地側のこの陸路に面した側にガードタワーを一列に建てる。陸路をガードタワーで遮断するイメージだ。ここに炎の戦士を主に住まわせる。ここまでが第1段階だ。

そして、このガードタワーの中の炎の戦士の射程距離をワールドビューで確認してほしい。各ガードタワーのアイコンを中心に自軍カラーの円が描かれていると思う。これがそのガードタワーに登った炎の戦士の「ブラスト」砲撃の射程距離なのだ。

この範囲内の陸路上に、もう一列ガードタワーを建て並べ、今度はここに伝道師やスパイを住まわせるのだ。これが第2段階。

これにより、互

最初から使えるスペル	
火山噴火	ライトニング
死の天使	土地隆起
ファイアストーム	マジックシールド
浸食	透明
地震	ハチの大群
土地平坦	幽霊軍隊
腐食	コンバート
トルネード	ブラスト
催眠	

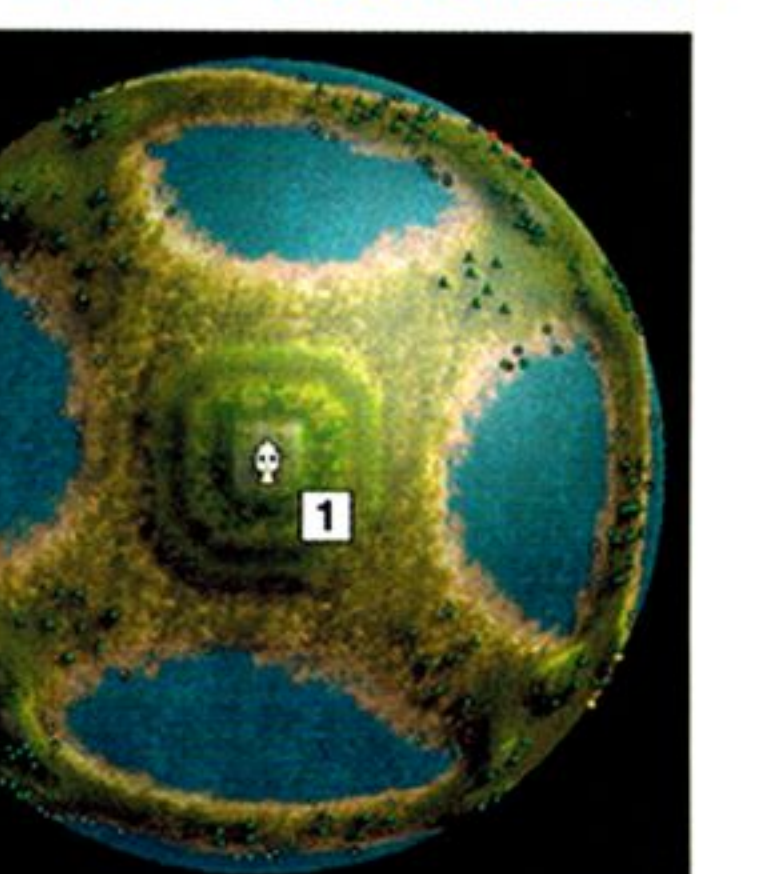
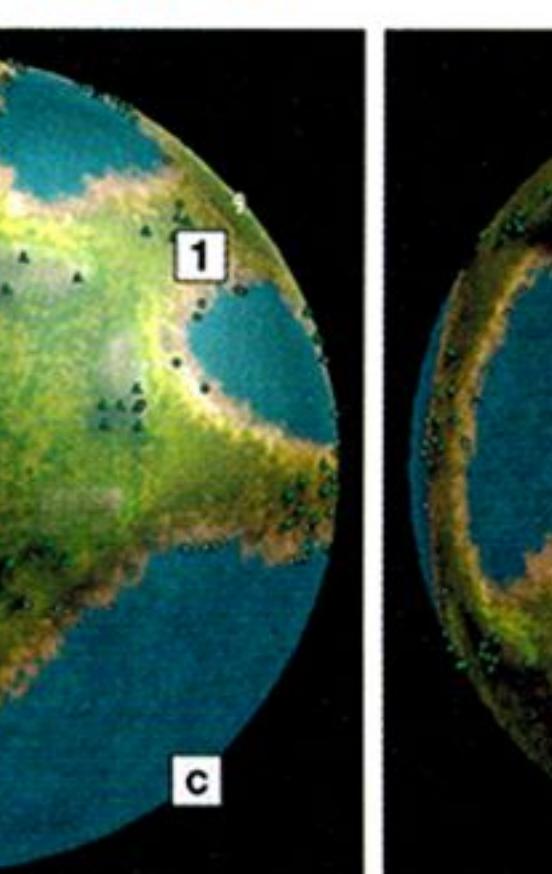
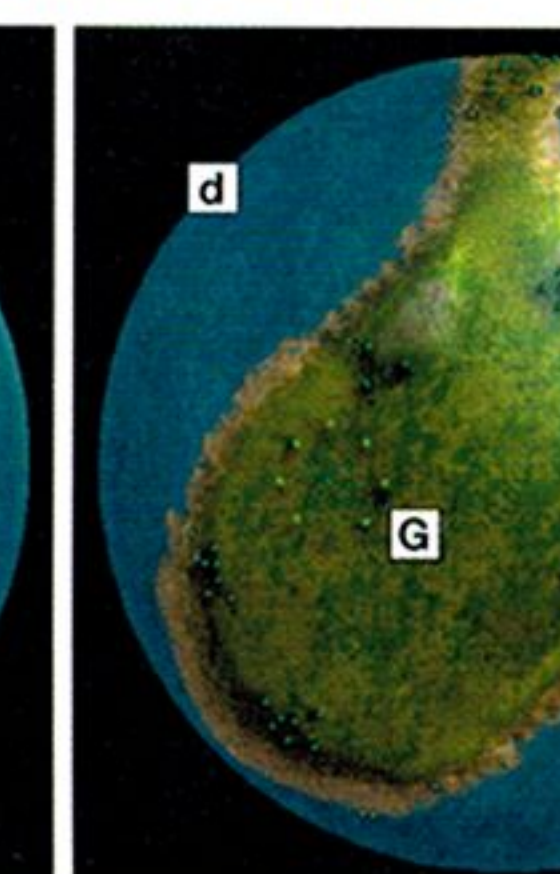
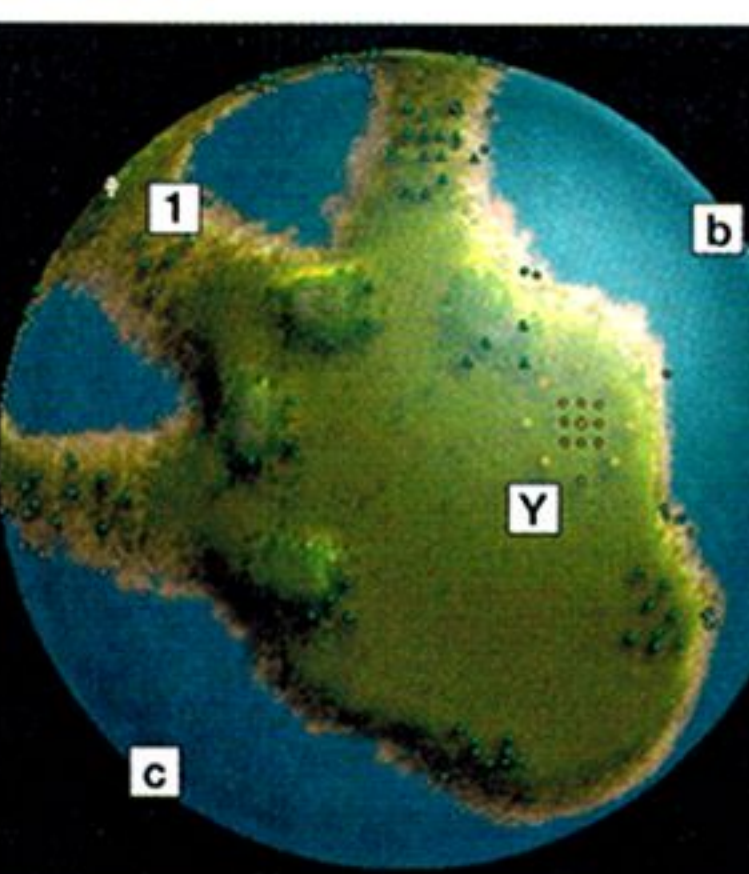
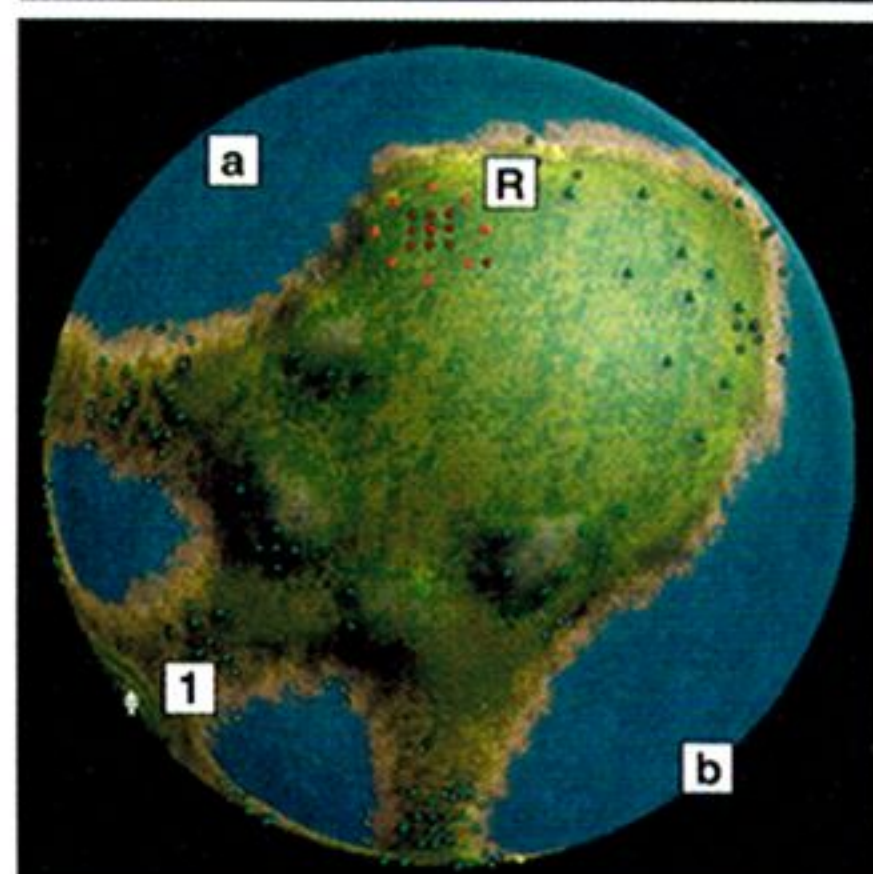
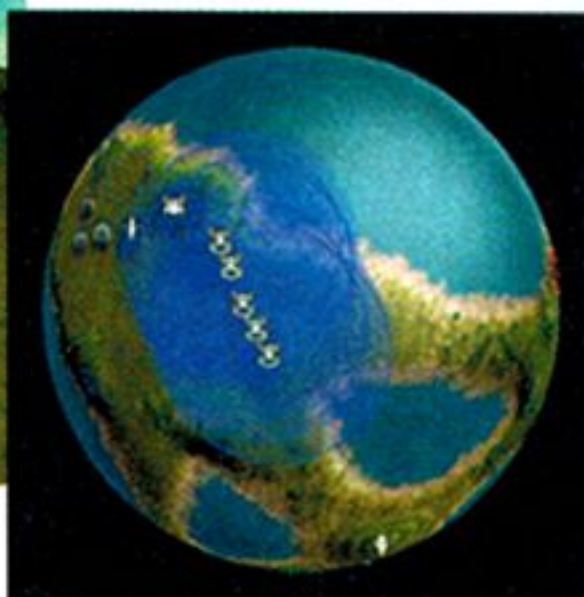
最初から建てられる建物	
ガードタワー	スパイ訓練小屋
小屋	気球小屋
伝道師訓練小屋	ボート小屋
戦士訓練小屋	ガードポスト
炎の戦士訓練小屋	

このレベルの概略図
1. 「ハルマゲドンスペル」(×1)

ゲーム開始直後にシャーマンを失った軍勢はしばらく、野人のコンバートができないことになる

互いの陣地を結ぶ陸路上に野人がたくさん群れている

ぼーっとしている敵シャーマンを見かけたら、

「ブラスト」スペルを放ち、池に沈めてしまえ。スペルを放つ方向に池がくるように、シャーマンを回り込ませてから撃つと成功率が上がる(ちょうどこの写真のような位置関係)

防衛ライン第１段階の守備範囲。陸路上のこの円内に第二段階を構築するのだ

防衛ライン第１段階。写真では手前側が自軍陣地

後列の第１段階防衛ラインの射程距離内に，前列の第２段階防衛ラインをいれるのがコツ

第２段階完成。画面奥側の方には伝道師やスパイを住まわせておくといい

いの防衛ラインの苦手な部分を相互に補いあうようになり，より強力な二重の防衛ラインが完成するのだ。

たとえば，敵戦士が侵攻してきた場合は，防衛ライン前列の伝道師がこれを洗脳して陣地内への侵入を食い止めてくれる。敵伝道師がやってきた場合には，前列の防衛ラインの伝道師と格闘戦が始まるが，後列の防衛ラインの炎の戦士がブラスト砲撃して援護するので，まず，負けることはない。

これを３本ある陸路，全部において完成させてしまえ鉄壁の防衛ラインが完成する。木々が豊富なこのレベルならではの作戦といえるだろう。

ストーンヘッドはどうするべきか

各陣地に囲まれた格好でストーンヘッドが小高い丘の上に立っている。

これは「ハルマゲドン」スペルを与えてくれるものだが，はたして礼拝すべきものなのだろうか。

礼拝時間は６人で礼拝して約５分30秒もかかるため，その間にさまざまな妨害行為を受けることは必至，礼拝の成功率はきわめて低くなる。特に同盟なしの４勢力が互いに敵対している状態では，このストーンヘッドの礼拝に気づくやいなや，全敵軍がこのときだけ同盟したのかと思えるほどの激しい妨害作戦を打って出てくることだろう。

ここは，「敵に与えるくらいならば，取られないほうがマシ」というふうに考えて，「腐食」スペルが数発ストックされたら，このストーンヘッドの前に重ね置きし，誰も礼拝できないようにしてしまったほうがいい。

もしこのレベルを，「同盟あり」の２対２のチームプレイでプレイしているならば，２つの軍勢が力をあわせて，ここの礼拝の完遂を試みるのも面白い。このストーンヘッドに近づく敵軍を５分30秒間撃退し続けるには息のあったチームワークが要求されることだろう。

「透明」スペルと「マジックシールド」スペルの両方をかけた伝道師は……

鉄壁の防衛ラインをいとも簡単に突破できてしまう

侵攻作戦・その１

野人と木々が豊富なので，どの軍勢も防衛ラインを整えるのは早い。大軍勢で正面から攻め立てても，大損害を被るだけだ。侵攻にはちょっとした工夫が必要になるだろう。

もっとも単純で，効果がそれなりに期待できるのは「透明」スペルと「マジックシールド」スペルの両方をかけた従者を敵陣へ進行させる方法だ。

この場合，防衛ライン上の炎の戦士の目は完全にかいくぐることができるので，敵陣へは簡単に潜り込むことができる。

ただ，伝道師以外の従者は，たとえ透明であっても，敵の伝道師が住んでいるガードタワーの横を通り過ぎた瞬間から，その声に耳を傾け始め，洗脳されてしまうので，結局，この方法で侵攻させてちゃんと役目を果たせるのは伝道師だけということになる。

その「透明」&「マジックシールド」をまとった伝道師を敵陣に侵入させたら，なるべく１カ所に固まらせないで，敵陣内で分散させて，敵従者を洗脳させること。

「透明」スペルは敵従者と遭遇すると，その効果が切れてしまうが，「マジックシールド」スペルの効果があれば，「催眠」スペルや「腐食」スペル以外のスペル攻撃は跳ね返してしまうので，敵プレイヤーには相当なプレッシャーが与えられる。

侵攻作戦・その２

防衛ラインそのものを破壊してしまう作戦も考えてみよう。

通常はシャーマン自らが出撃し，「ライトニング」スペルなどでガードタワーを排除していくことで，防衛ラインに穴を開けていくわけだが，このレベルは木々が豊富でその防衛ラインを構成するガードタワーの数が多くなりやすいため，なかなかこの方法で穴を開けるのは難しい。

そこで，この「防衛ラインの破壊」を死の天使に任せてしまおう。「死の天使」スペルを敵防衛ラインの手前で発動するだけでいい。死の天使はガードタワーの中の炎の戦士から優先的に襲ってくれるはずだ。

あとはこの機能停止した防衛ラインから，一斉に大軍勢を送り込めばいい。

侵攻作戦・その３

ほとんどのプレイヤーは敵が，自軍陣地から伸びている３本の陸路からしか侵攻していないと思い込んでいるはずだ。だから，防衛ラインも，この陸路からの侵攻対策用のものしか構築していない場合が多い。

そこで，ボートや気球といった乗りものを駆使し，この３本の陸路を使わずに陣地の背後，海のほうからの奇襲をかけてやるのだ。

敵防衛ラインに十分近づいてから「死の天使」スペルを発動

死の天使はガードタワーの中の炎の戦士を優先的に襲っていく。このときガードタワーも１段階破壊されるので，防衛ラインには穴が開くことになる

取られるくらいならば取れないようにしてしまえ，ストーンヘッド

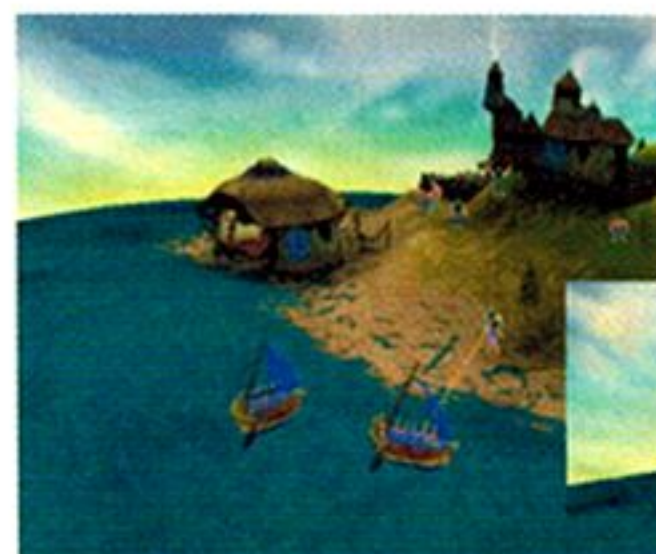
連係プレイを心がけて５分30秒間ストーンヘッドを死守せよ

「ハルマゲドン」スペルを入手したら，速攻で戦士の大量育成を開始しよう。ハルマゲドンは戦士の数が多ければ多いほど勝利する可能性が高くなる

このレベルではボート，気球の両方が作れるのでこの両方を駆使してみよう。

たとえば，こんな作戦が有効だ。まず，「マジックシールド」スペルをかけた伝道師を，ボート２，３隻で海から敵地へ上陸させ，続いて同じく「マジックシールド」スペルをかけた炎の戦士の気球軍団を敵地の中央付近に停泊させるのだ。

上陸したこちらの伝道師は，敵地の戦士，炎の戦士，勇士の洗脳を開始するはずだ。これに対して，敵は，当然この洗脳活動を妨害するために，自分たちの伝道師を移動させてくることだろう。それを迎撃するのが，上空で停泊している炎の戦士の気球軍団というわけだ。

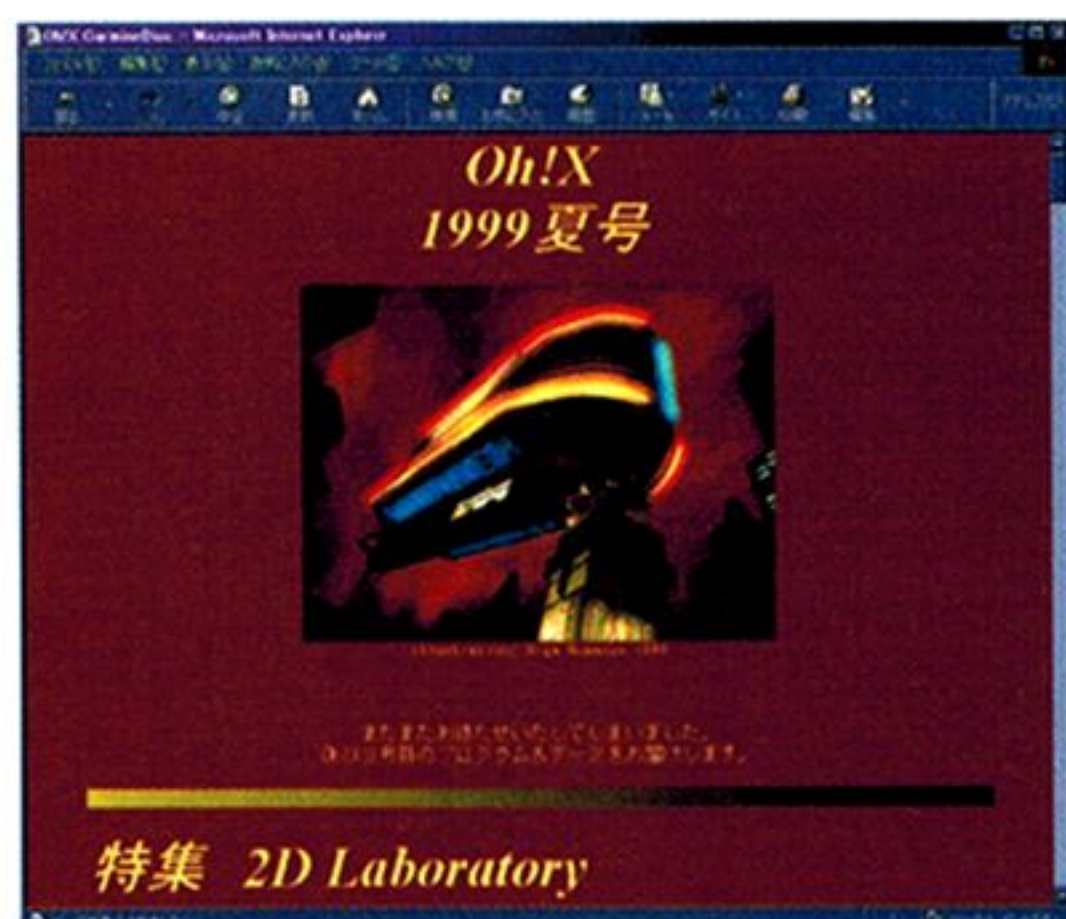
攻められた側としてはシャーマンをこの戦域に移動させ，気球軍団や伝道師に「催眠」スペルを使って状況を打開しなければならなくなる。攻め立てる側としては，敵シャーマンに「催眠」スペルをかけられては作戦が水の泡と化すので，敵シャーマンの接近を察知したら，早急に全気球軍団の攻撃目標を敵シャーマンに定め，一斉砲火を浴びせて撃退しなければならない。

「マジックシールド」をまとった伝道師をボートに乗せて，上陸作戦を開始する

続いて，気球軍団にも「マジックシールド」をかけて，これを敵地に移動。先鋒の伝道師部隊の護衛に回るのだ

攻められた側としては「腐食」スペルか「催眠」スペルでしか反撃方法がないことになる

付録CD-ROMの使い方

これまで2枚組だった付録CD-ROMですが，今回は1枚のみ。掲載プログラム＆データを中心に構成されています。X68000用のプログラムもなるべく多めに収録してみました。

CD-ROMのメニューはHTMLで記述されていますが，Webブラウザなどは搭載されておりません。各自でご用意ください。シフトJISコードによるテキストファイルですので，Webブラウザをお持ちでない場合でも最低限の内容確認は可能です。面倒なら直接ディレクトリ内を参照してください。まれにシフトJIS以外の文字コードの部分がありますのでご注意ください。
以下，誌面で紹介しきれなかったものを中心に解説します。

X68000用ツール

●Z-MUSIC ver.3.0用ツール
じんべさんの投稿によるZ-MUSICver.3.0用のツールです。SMF→ZMDコンバータ"ZMC_SMF.x"、RCP→ZMDコンバータ"ZMC_RCP.x"、シンプル＆セレクトプレイヤ"SSZP.x"、MIDIメッセージレコーダ"ZMR.x"、インフォメーションインジケータ"ZII.x"、メモリインフォメーション"ZMEMINFO.x"、SMF→ZMSコンバータ"SMFtoZ3S.x"、MDX→ZMSコンバータ"MDXtoZ3S.x"が収録されています。

●CODEAドライバおよび辞書ツール
16Mバイトまでの辞書に対応させたASK68K互換のFEP、Code name ASKの開発途中バージョンです。CODEAおよびASK68Kの両方の辞書に対応した辞書ツールも含まれています。

●ANOS満開製作所制作ゲームデモ
満開製作所が開発中のX68000用新作ゲームのデモバージョンです。詳しくは　ページを参照してください。FM音源による高音質PCM再生などに注目。メインメモリ4MB以上必要です。

●DuelFighter2
1999春号のUSER'S WORKSで紹介したチーム無限うなぎ制作のシューティングゲームです。多少のCPUパワーを必要とします。

●ProtePruste
　ページで紹介しているシューティングゲームです。作者の環境がX60000/10MHz（2MB）なのでご心配なく。

●Linux
X Window上でPlayStationやSEGA SATURNのムービーファイルを再生するためのツールです。1999春号で収録したソフトのバージョンアップ版です。

特集関連

●デジタル入校原稿
マンガ原稿のデジタル入校はどの程度現実的かということで，今回実際に使用されたデータです。レーザープリンタをお持ちの方は、どの程度の出力が出るか試してみるのもよいでしょう。

●VMAKER体験版
スキャン画像などのビットマップデータからベクトルデータを取り出すためのツールです。ページの解説も参照してください。
提供　アイフォー

●GIMP
GNUのグラフィックツールです。X Window版とWindows版の最新安定バージョンが収録されています。それなりに重いので注意。

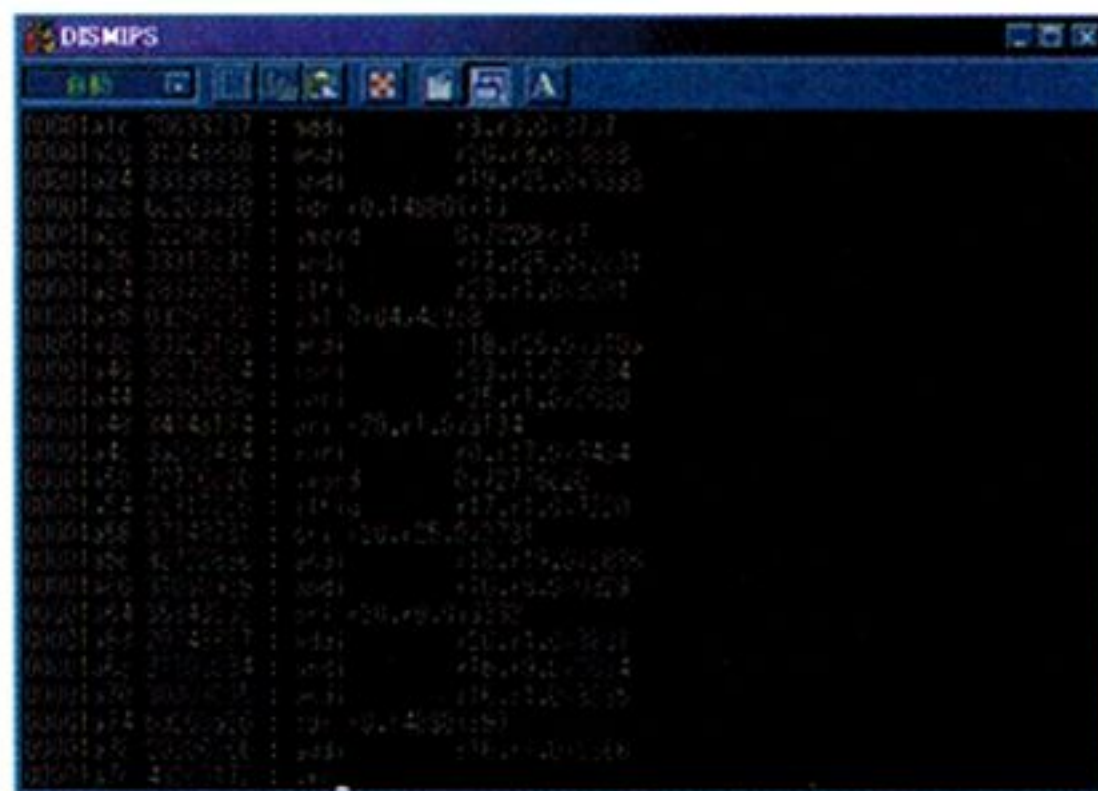
●EX-System for Win
グラフィックツールEX for Win用のプラグインフィルタの作成法を解説します。サンプルはメディアンフィルタです。

Oh!X LIVE in '99

Final Chance（カシオペア）
作曲　向谷実
データ作成　マッチュン
日本音楽著作権協会（録）許諾R-9970129

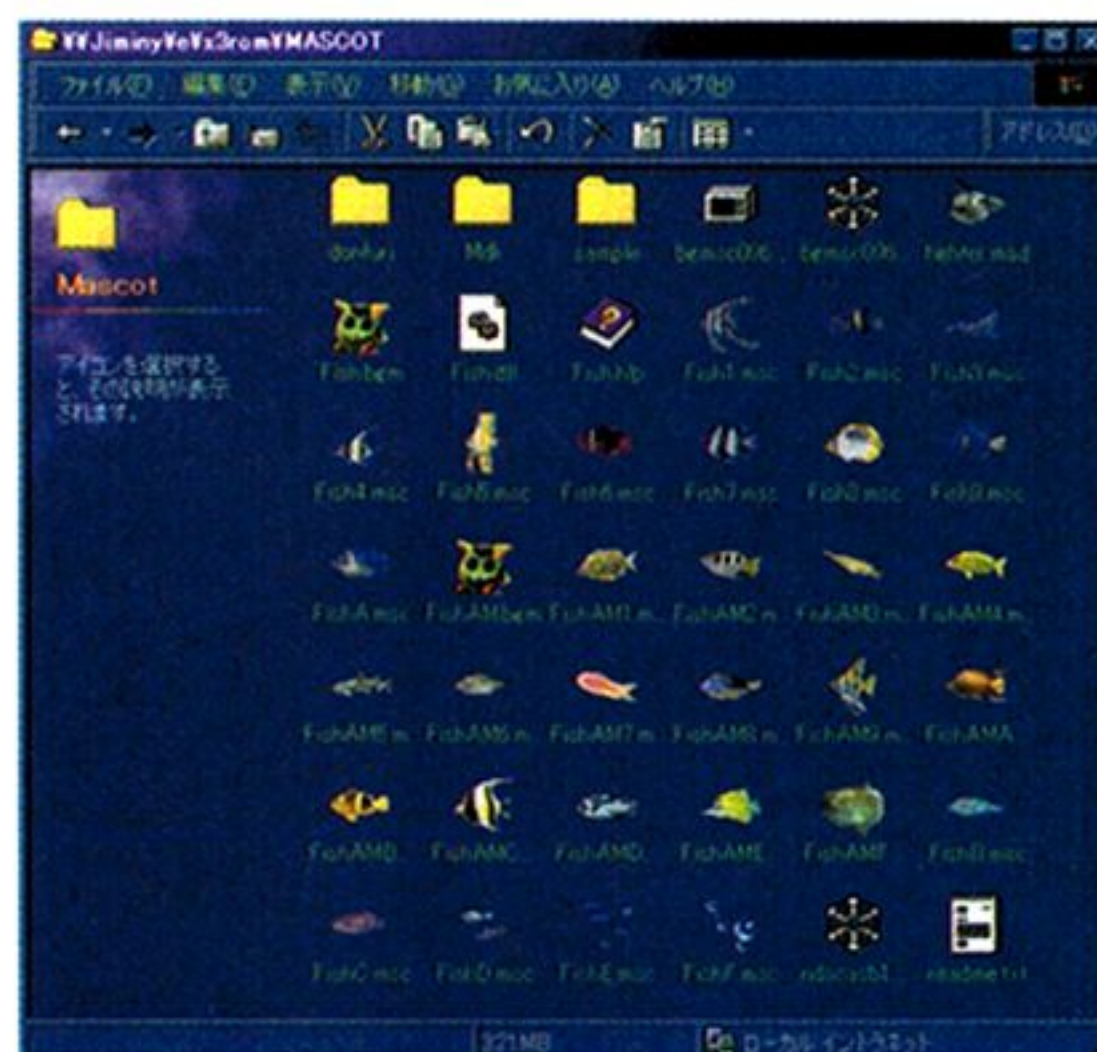
音声トラックとZMSデータの形式で収録されています。

その他のツール

●立体表示用プログラム
Direct3DのXファイルビュアのステレオ表示対応バージョンです。内容は前号に収録されたものと変わりません。

Rシリーズ用逆アセンブラ

●MIPS16/32/64対応アセンブラ/逆アセンブラ（各機種）
MIPS RシリーズCPUの大半の範囲をカバーするアセンブラ/逆アセンブラです。添付オブジェクトはx86機種用（Windows）EXEファイルです。プレーンなCプログラムですので，機種は問いませんが，逆アセンブラ部はネストが非常に深いので，そのままではコンパイルできない場合もあります。各自で調整してください。

●オリジナルPICアセンブラ＆PICシミュレータ
プログラム制作　高尾克彦
サンプルプログラム制作　石上達也
PICマイコン用のアセンブラおよびシミュレータプログラムです。

●Background Mascot Shell & Mascots
プログラム制作　菊地功
データ協力　どんぶり
Windowsのデスクトップ上でさまざまなキャラクターを動かします。Windows95＋IE4の場合は画面が正常に更新されないことがあります。その場合はActiveDesktopを設定するか，背景に壁紙を使用してください。

Mascot Shell

エロゲーにエッチは重要？ 物語のリアル論

古村聡 Komura Satoshi

さて，またまたマルチで申しわけないが，非18禁のPlayStation版発売記念ということで，Windows版と比較してエッチゲームにおけるエッチの必然性と，それがストーリーに与える影響について考えてみよう。

ご存じのように，今年（1999年）の春に無事にPlayStation版のTo Heartが発売されました。中身は……元の18禁ビジュアルノベルだったWindows95版がベースなんですが，ご家庭向けゲーム機では許されさなそうな表現（早い話がエッチシーンだね）を削り取って，ほかの話に入れ替えたもの，だと思えばだいたいあっています。

一応，それ以外にもWindows版で評判がイマイチだった志穂シナリオのように，シナリオ自体が変わってしまったとか，せっかくのゲーム機だからというわけなのかアクションやシューティングなゲーム内ミニゲームが追加されたりといった違いもありますが。

ええ，やっぱりというか当然というか，私も買いました。PlayStation版To Heart。ちなみにWindows95版もなぜか2つ買っていたりするので機種は違うものの，合計3枚目のTo Heartだったりします。ああ，貧乏なのにやっぱり消費させられてしまう，オタクの業の深さよ。

で，ですね。当然のように，まずはPlayStation版マルチシナリオをプレイしてみました（編注：複数のシナリオという意味ではありません。マルチのシナリオです）。

ところがですね。どうもいけないんですねぇ，これが。オリジナルのマルチのシナリオは「夜中に家に訪ねてきたマルチだったが，もう帰らなければならないというときに，主人公・浩之との泣かせるやりとりのあと，一晩だけ主人公の家にとどまって，結果，ふにふに，あああっ」という，感動のエッチ込みストーリーが展開されるわけですが（詳しくは本誌復刊1号参照），PlayStation版では，夜中にマルチは訪ねてきません。翌日昼です。どういうわけか遊園地なんかに行ってしまいます。エッチシーンに代わるのは2人でデートして，観覧車に乗って語りあうシーンです。

「今日のことは，メモリー内のもっとも大切な部分に記憶しておきますから……」

そして楽しく（プラトニックに）時間を過ごし，そしてマルチは研究所へと帰っていきます。でもってこのあと，PC版と同じように再会シーンへと移るわけです。

このPlayStation版のTo Heart自体は，評判は悪くありません。それどころか「PC版よりこっちのほうが好き！」という意見さえもときどき聞かれます。けれど……，マルチのシナリオに関しては残念ながら，どーも，なにかもの足りないんですよね。そうは思いませんか，プレイされた方。私なんぞは，PlayStation版をプレイしたあとで，もう一度，PC版のほうをプレイし直して「ああ，よかった。うんうん，やっぱりこっちが本当のマルチシナリオだよねぇ」と再確認作業してしまいました。

なんで，こんなにもの足りないのでしょうかね。エッチシーンがなかった，というだけの話で。

愛だろ，愛っ

結論から簡単にいってしまいましょう。ずばり，もの足りなく感じるのは，マルチシナリオをはじめとしたストーリーたちで語られているのが「愛」だからなのです（うわぁ，恥ずかしい。いうな，いってる私がいちばん恥ずかしいんだから）。

考えてみると，エッチゲームといっても，昔は，わりと「裸が出ればそれでいいだろう」というようなゲームもあったんですよね。たとえば，ヌードの絵が16パズルになっていて，それを見るだけとか。女の子が脱ぐだけの脱衣マージャンというのもこの類に入るかな？　とにかく，女の子が脱ぐのは単にゲームを進めるうえでの「エサ」である，というタイプの使い方でのゲームです[*1]。

しかし，時代が進むにつれて，エッチシーンのあるゲームでは，エッチシーンの前後に「愛」を語るようなゲームが多くなってきます。

たとえば，これもかなり古いゲームですけど，FM-7などにあった「東京ナンパストリート」では，ナンパして女の子をホテルに連れ込んでエッチしたあとは，必ず，その女の子と恋愛関係になって，結局その子と結婚する，というエンディングを迎えるんですよね。

「ゲーム中にエッチがあるんだから，愛とか恋がないとまずいだろー」という作者側の良心からなのか。そして，いまでは，「愛」がストーリーの核になって，現在の主流はゲーム中の女の子たちと恋仲になって，愛情が高まった状態で，愛を確かめあうようにエッチシーンに流れ込むパターンの18禁ゲームが多くなっていったわけです。というか，考えてみると，表現方法はいろいろありますが，いまではストーリーで進めるタイプの18禁ゲームってだいたい「愛」を語るゲームですよね。

では，オリジナル版でのマルチのストーリーがどうやってエッチまで流れていったかを思い出してみましょうか。まず，主人公は学校でマルチというメイドロボットに出会います。主人公は，マルチのイノセンスと，純粋な「人間のお役に立ちたい」という夢を持つピュアさにひかれていきます。そして，マルチとの別れの日がやってきます。

「…わたしはもともと試作機です。ですから、この試験期間が終わると同時に、その役目も終わりになります。だから、浩之さんとも、今日が最後になります」
「…やっぱり、そうなのか？」
「…はい。黙っていて、すみませんでした」

「…今日は、とっても幸せでした。わたし、あんなお家で、浩之さんみたいなご主人様のために働くのが夢だったんです。今日はそれが、短い間ですけど叶いました」

PlayStation版のマルチシナリオ。基本的な部分は同じなんだけど……

「…わたしはもともと試作機ですし、この試験期間が終わると同時に、その役目を終えるんです。だから、浩さんとは、もう会えなくなるんです」

「今日はとっても幸せでした。わたし、こんなお家で、浩さんみたいなご主人様のために働くのが夢だったんです。今日はそれが、短い間ですけど叶いました。

こちらはWindows版マルチシナリオ。同じ台詞なんだけど，エッチがあるかどうかで全然効きが違うよね（と思うんだけど）

マルチはいままでのお礼にと主人公の家にやってきます。主人公はマルチとの会話で，マルチがもう，二度と主人公とは会えないこと，彼女の魂ともいえる，彼女の記憶を持った彼女の心が二度と倉庫の奥から出られなくなることを知ります。

さて，この時点でプレイヤーは相手，つまりゲーム上のキャラクターに感情移入していますから，マルチも自分のことを愛してくれているといいなと思います。２人の絆を確認する作業がしたい。そして，彼女が自分を受け入れてくれるのなら，２人がひとつになりたいと。２人だけの場所で２人だけでお互いを感じたいと。

キミがほしい＝エッチがしたい

という流れでエッチに流れ込んでいたわけですね。

マルチ側も「私も浩之さんのようなご主人様がほしかったのです」と主人公への愛を言葉にしつつ，主人公を受け入れてます。

最近ですと，Kanonの栞シナリオなんかもそうでしょうか。……感動したなぁ。病弱で学校に出てこられない少女と主人公は出会うんですよね。栞は病気で次の誕生日まではもう生きられないといいます。なぜか彼女の姉は，まるで彼女が存在しないかのようにふるまいます。ああ，繰り広げられる，姉妹愛，そして，主人公との愛の物語……さらに，愛が盛り上がったところでエッチシーンが繰り広げられます。ううっ，感動的だよぉ（詳しくはご自身でプレイしてみてください）。

ま，人間の心理状態を考えれば，これで当然，なんですが。人間というのは，自分の愛するものに受け入れてもらいたい。でも，愛するものは自分以外という存在。男と女という違う存在。生い立ちも考え方も，物理的にも違う存在。受け入れてもらう存在と受け入れる存在。とても違った存在なのですよ。だから，男女の愛には，その絆を確認するための作業が必要なんです。それが恋愛での「結ばれる」感覚なんです。これはゲームに限らず，男と女の物語である以上，お話であるものには絶対に必要なものだと思うんですよね。

さて，翻って，エッチなしのPlayStation版のマルチシナリオを見ているとひとつのことに気づきます。そう，先ほどの「観覧車シーン」なのです。エッチがなくなってしまっただけでなく，そこで行われた「２人がお互いの気持ちを本当に確認しあって，絆を確かめあう」というプロセスがなくなってしまっているんですよね。

デートに行けた。２人きりになれた。でも，そこまで。お互いの心を確認する作業がない。だから，その先のマルチがいなくなったときの喪失感が大きくない。そのために，最後のマルチとの再会したときの感動が物足りなくて，全体にエッチシーン以降のストーリーがぬるく感じられてしかたなくなってしまうんですよね。だから，本当のところは「エッチがなくてもの足りない」ではなくて「２人の絆を確認するイベントがなくてつまらない」なのかもしれません。

でも，さて，男女２人の絆を確認する手段で，かつ２人の主観でインパクトが強いのはどんなことがあるでしょう？　まぁ，心中とか，駆け落ちなんていう手段もあるかもしれませんが，平和的な手段では，やっぱりエッチですよね。だから愛を語るゲームにはエッチシーンがちゃんとあること，というのは重要なんですよね※2。

ということで18禁ゲームのHシーン描写にもの申す

ただ，現存する18禁ゲームのエッチシーンを見ていると……「２人の絆を確認した」といえるほど感情移入できるものって，なかなかないのも事実なんですが。ひと言でいうと，まだデキがイマイチのものが多い，というか。たとえば，こんな感じで興がそがれてしまうんですよね。

・うしろめたい気分にさせられてしまう

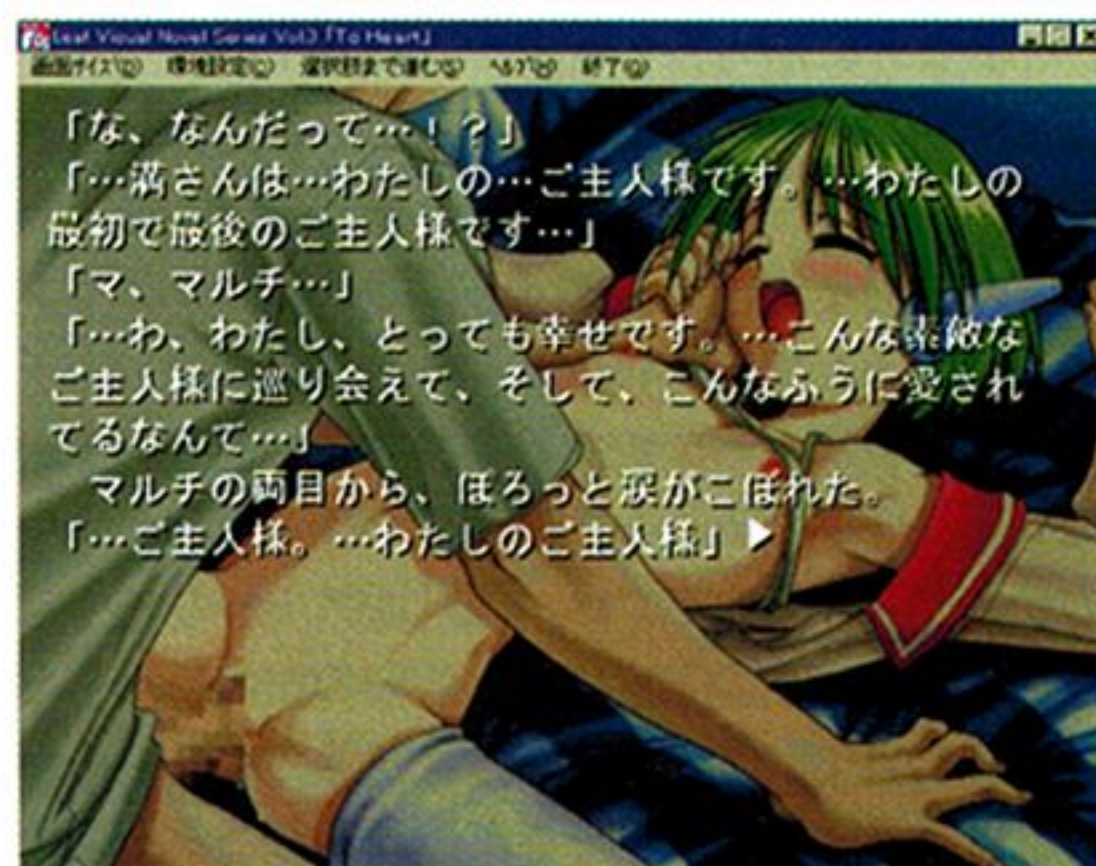
話の盛り上がり方がエッチシーンに向かないために，エッチシーンになると後ろめたい気分になって，やる気をそがれてしまうケース。たとえば「無理やり」とかですね。あるいは人の弱みにつけ込む，とか。

最近の18禁ゲームでときどきあるのが「癒し話」とでもいうんでしょうか，女の子の心の傷，たとえば，実は本当に好きだった彼が死んでしまったのとか，たったひとりの身寄りも死んでしまって私はひとりぼっちなの，とか。そういう孤独感にさいなまれている寂しい心を慰めて，ついでにいただいてしまう話なんていうのが典型じゃないかと思います。

心理的なカウンセリングをやっている方に聞いたことがあるんですが，そういうときは文字どおり親身，つまり親や肉親の代わりと思える状態に誘導して話が聞けるようになるんだそうですね。で，そこで生まれた親密な感情を「恋愛感情」と間違えてしまう危険なケースも確かにあるんだそうなので，まぁ，お話じゃなくて実体験としてそういうケースがある場合もあるのかもしれませんが。

※1　過去のエッチゲームを思い出してみると…その頃（で）って18歳未満だったので実はよく知りません。単に「愛がなさそう」ってのは「女子大生プライベート」とか「団地妻は電気ウナギの……」とかの雑誌広告を見てそう思っていただけです。ですんで，ここのところは間違えているかもしれません。同じころに「ファイナルロリータ」とか，あったはずなんですが，あれはどういう話だったんだろうなぁ。ちょっと不安。

※2　余談：逆にいうと，それが不幸であれ，幸せであれ，エッチに代わる大きく心を揺さぶるようなイベントが続くストーリーなら，エッチがいらない，ということも当然ありうると思います。その意味ではエッチがいらないストーリーテリング的18禁ゲームというのがある，というのもわかります。よくいわれるONEはやったことはないのでわかりませんが，Kanonにはそういうシナリオもありますね。確かに。

PlayStation版の山場……って，抱き締めといてそれだけか？

Windows版はこうやって山場（濡れ場？）に突入するわけですね

深い絆があってこそ，別れのシーンが印象深いものになってくるわけです

Kanonより栞シナリオ。緊迫した姉妹の関係。それぞれの愛情にまつわるストーリーが繰り広げられます

・話の流れがエッチシーンで変わってしまう

よく18禁ゲームでエッチシーンがそこだけ突然「ブチっ」と切れたように話が変わって邪魔に思えてしまうというパターン。たとえば，それまで女の子にやさしくしていたのに，いきなりオノレの欲望に忠実になって，ムリヤリしちゃうとか。

まぁ，どんなにがんばって見せても，エッチするときには結局自分の欲望を満たしたいってエゴは隠せませんから，現実だって似たようなものなのかもしれないんですが……本人は据え膳を目の前にしたら，そっちを食うのに夢中になって自分がいまどんな状態なのかは見えませんもんね。本当はどうなのか，実際食べられちゃう側の女性からみてどうなのか聞いてみたいような……(聞かないほうがいいような気もするけど)。

・変な描写

こんなエッチするか？　と思うような変な描写です。いや，たとえば，初めてのエッチなのにいきなり後ろから……しちゃうとか……ですね。

普通に考えたら，最初っから，後ろからはないでしょう，いくらなんでも。男だって後先考えますから「こいつ，いまだけ遊んで捨てちゃえぇ。ポイだ」とでも考えない限りしませんて。女の子だって，その男とは初めてだったら恥じらいもあるからいいませんって。万が一，本当はそっちのが好きだったとしても(←ううっ，もうなんか，書くの恥ずかしくてしょうがないんですけどー。ここまで書いちゃったらトコトンまで。ええ)。

だいたいにおいて，一般に18禁ゲームって(私がプレイした範囲内で，ですけど)，エッチ描写が(物理的な描写も，心理的な描写も)全然リアルじゃない気がするんですよね。やっぱり，気合の入れ方が足りないような気がするんですね。それとも書き手のテレなのか。エッチ描写でありがちだよなー，とよく読んでいて思うんですが，「いじる」「濡れてくる」「ああっ」という声がする，入れる，気持ちよがる，ってルーチンワークのように進んでいく。こういうのを読むとなにか，エッチってこういうものだったっけ？　なんかリアリティというものが全然感じられないよなー，というような事態に陥ってしまうのですよね。

物語のリアルさっていうのは

さて，ここでふと思うのがリアリティについてなのです。リアルとは「現実っぽい」という意味ではない，と思うのですよね。

たとえば，話中の時刻がいちいち書かれているとか，女の子のアレが30日よりちょっと短いくらいの間隔で話中に織り込まれているとかではないですよね(どうも今回の原稿はこういう方向に走るなぁ)。ストーリーにとってのリアリティとは，いったいなんなんでしょう。先ほどの「リアルに感じられない」例には，どれにも，共通点があるんです。それは「ストーリーが破綻しているのがわかってしまうこと」なんですね。それまで，2人の愛を確かめようとしていたのに，急にどれも，ここで辻褄があわなくなっているんですね。しかしですね，どんな物語も人間の作ったものですから，矛盾，あるいは破綻というのはある意味宿命です。つまり，どんな物語も「リアルでない物語」に転落してしまう可能性を秘めています。必ずなんらかの矛盾があると考えてもいいです。なんとかして矛盾点を見逃してもらうためにはどうすればいいか。その方法としてよく使われるのが「プレイヤーの注意をほかの点に引きつける」ことなんですね。

関係ない話ですが，なにかの特撮ヒーローモノでは，そのヒーローが空を飛んでいくシーンで人形をピアノ線で吊るしているのを気づかれないようにするために，ヒーローの人形を腹側から逆さに吊るして，カメラも逆さにして撮影したんだそうですな。つまり，子供たちが背中から吊るしてある糸を見つけたらがっかりしてしまうでしょう。子供たちは背中から天井に向けて糸を探そうとするでしょうから，盲点になるように，逆に腹から吊るしてぶら下げておいたんですって。

人間，だまされまいとして一点に注意しちゃうと，かえってほかの手で簡単にだませてしまうものだったりするのですね。つまり，ストーリーにリアリティを感じられるかどうかは，プレイヤーの「感情移入」をどこか一点に向けさせ，いかにそこから放さないか，にかかっているわけです。

で，話は戻って，18禁ゲームのシナリオですが，18禁ゲームをプレイして涙しちゃうというのも，もちろん，「主観的リアリティ」あってこそ。こちらはプレイヤーの目をよそに向けさせるといっても，模型の吊るし方を工夫するわけじゃなくて，話運びでプレイヤーを感情移入させてるわけです。ストーリーを進めていくうえで，見る人に感情移入させる定石っていうのはいろいろあり，よく知られてますね。たとえば，

- キャラクター(女の子)をどうしようもないような境遇において，プレイヤーを同情させる
- とんでもなく強いキャラだったのにある事件が起きてから実はとてももろいパーソナリティだったことがわかる

とか，です。いわゆる「お約束」というやつですが，それらを使ってある程度までプレイヤーがゲームのキャラクターやストーリーに感情移入したとしましょう。で，大事なのはここから。

ここで，重要なのはこれらのテクニックを駆使しつつ，自分の視点を，感情をゲーム中の世界に投影するのに邪魔をさせないようにすること，なのです。物語というのは，そのなかでひとつの閉じた異世界を作っているわけですが，優れた物語を作ろうとしたら，現実世界を導入に使っても，プレイヤーをエンディングまではそこから放さないでほしいのですね。たとえ18禁ゲームでも。

具体的にはどうするかというとたとえば，先ほどの特撮の吊り線ではないですが，プレイヤーが物語世界以外のものに直面することがない。感情移入の邪魔になるものがないようにするわけです。それは，エッチのときこそ，男の神経はとぎすまされて真剣勝負になっているということを思い出して書くのです。ほら，エッチのときって，相手の表情を結構見ているでしょ，それを書くんですよ。ほら，汗かいてくるじゃないですか。それから筋肉の1つひとつが引っ張られるような感じとか，見ててわかるでしょ？　女の子の表情や仕草から感覚やしたさの有無を瞬時に判断して反応を返す。そういうコミュニケーションが本当はあるはずなのに，ないんですよね。ゲームストーリーには。それまでは心が通ったような会話をしていて，そこだけ感覚が一方通行じゃだめ，やっぱり，ここも双方向じゃなきゃいけないと思うんですよね。身も心も。

やっぱり，手順はまずジャブ。相手の様子を見る。相手の力の入り方，筋肉のこわばり，表情から相手の心を見切って，そして相手のあごに正拳突き！じゃなくて，ふにふにする心あるエッチシーンを提供していただきたい！　というわけで，私は提言したいのです。エッチシーンを書く者はもっとエッチシーンは2人の身になっててていねいに心を込めて書け！　ということをです。いや，エッチシーン，恥ずかしくて書きづらいのはわかるんです。こんな話を書いている私だってすっげい恥ずいんスから。でもですね，やっぱりちゃんとエッチシーンも心込めて書かないとシナリオそのものがそこでおかしくなってしまうのですよ。愛にエッチは重要ですけど，エッチにも愛が重要なんですよね。

もちろん，基準がプレイヤー1人ひとりの主観ですから，ある人にとってリアルであっても，ある人にとってはリアルでないこともありえます。

たとえば，先ほども出てきましたが，「Kanon」の栞シナリオ。このお話では，雪の中で女の子が横たわるシーンがあるんですが（感動的なんだ，これが）「ああ，雪の中で2人……横になって……美しい」と思っちゃう人もいるわけです。でも，世の中には，雪国出身で「あんな雪が降っているところで横になったらすぐ体温が奪われて生きていられるわけないじゃん」と実体験から，覚めて指摘できちゃう人もいるんですよね。

この2人では感動のしかたは当然全然違ってしまいます。だから，万人に感動を与えたいのであれば，ある程度は事実を調べてシナリオライティングしなくちゃいけない，というのも事実ではあるんですよね。しかし，ある程度までやったらあとは事実にばかりこだわる必要はないのです。必要なのは感情移入。プレイヤーが，ゲーム中の自分に，女の子に，舞台に，世界にとにかく感情移入させて放さないように，がんばってほしいなぁ，と私は思うのですよね。

Image34　Kanonより，雪に横たわる栞。綺麗なシーンでねぇ，感動するんですか。雪国の人にいわせると「普通，凍死するだろう」です

男は顔じゃない「FINAL FANTASY Ⅳ」

野沢絵美 Nozawa Emi

FF Ⅳの画面。スーファミ出始めだったのでこれでも美しく感じた

キャラクターにほれ込むのもゲームの楽しみ方のひとつ。今回取り上げるFF Ⅳといえば，シリーズ最高傑作と讃えられるスクウェアの傑作だ。「なにをいまさらFF Ⅳ」といわずに，キャラクターとドラマ展開の典型的な成功パターンを見てみよう。

セシルが素敵

私が大好きなゲームのひとつにFINAL FANTASY Ⅳがあります。このゲームのなかでいちばんのお気に入りはなんたって主人公のセシル。ゲーム界の登場人物のなかでいちばんを選べといわれても迷わず彼を選びます。「己の理想の男性像をいえ」といわれるとやっぱり彼かな～。FF Ⅳは1991年にリリースになったゲームで，どーしてこんなに長く彼を愛していられるのかと自分でも不思議なのですが，どうやら最初に彼の人となりを知ったのが顔からではなく『心』からだったからのようなのです。

FF ⅣはFFシリーズ初のスーパーファミコン対応のソフト。私がソフトを購入したのは近所のスーパーマーケットです。その当時，品薄だったハードとソフトがおもちゃ売り場にあったので，いま買わないと今度いつお目にかかれるかとあせった私は，ネギを抱えた状態でハードとソフトを持って帰ったのです。

FF Ⅳはコマンド入力タイプのRPGゲームで，アクティブタイムバトルシステム（ATB）というのが新機能として謳われていました。ATBは，コマンド入力をしている間も時間が流れているから，のんびり操作をしているとダメージを食らっちゃうシステム。だから，コーヒーを飲みながら優雅にゲームをやっていた私の友達なんかには不評でした。それでもコマンド入力であっても実際に戦っている緊張感が持ててなかなか臨場感あふれる戦闘が楽しめました。FFシリーズがFF Ⅳ以降，ATBになったのもうなずけます。

ゲームシステムもさることながら，グラフィックも音楽（スーファミからステレオ対応になった）も一段とグレードアップして，このゲームのおかげで私が持つスーファミの第一印象がかなりいいものになったともいえちゃいます。

暗黒騎士セシル

ゲームスタート時に登場したセシルは暗黒騎士でした。暗黒騎士は暗黒パワーを剣技に応用して戦う剣士です。一般人は彼らを尊敬していますが，異様で不気味な暗黒の力を持つ彼らを恐れてもいます。暗黒騎士は暗黒系の剣と，暗黒系の鎧しか身につけられないため，見た目の印象がダークなうえに体格もすんごくごっついのです。顔だって不気味な兜で隠れて見えません。ダースベイダーとかハイランダーの敵とかいった感じです。セシルの容姿は主役というより悪役に近いのです。「FFシリーズの主役にしちゃあ不気味なワルワル系でごっついわね～」ってのがセシルに対する私の最初のイメージでした。

私は基本的に操作方法程度の予備知識しか持たない状態でゲームを遊ぶようにしています。じゃないと，ゲームの仕掛けとか，さまざまな発見を新鮮に楽しめないからです。だから，セシルはずーっと暗黒騎士のままでいるもんだと思いました。どっちかというと金髪さらさらの王子様が好きな私はその点がちょっと不満でした。ところが，ごついセシルの行動を追いかけていくと，年長者や恩人には礼儀正しく，部下や市民に慕われ，恋人に優しく，頼れる友人がいて，子供に人気があります。そのうえ正義感も責任感も人一倍強いのです。見た目はもの凄いのですが，性格と行動はちゃんと主人公している様子です。「見た目はアレだけど，セシルってば，性格とか行動はかっこいいので今回は許しちゃおう」とゲームを進めることにしたのです。

セシルはその後も幼い少女を救うために屈強の兵士たちを相手に戦ったり，自分を追いかけて病気になってしまった恋人のために魔物を倒して薬を手に入れたりと，さまざまな試練をクリアして

暗黒騎士のセシル。ダースベイダー様ってかんじでごつい。それでも愛しちゃう

パラディンのセシルの素顔。きゃ～，美しすぎる～。愛する度合が倍増

いきます。彼はごつい容姿（しつこい？）に反して，デリケートだったり，落ち込みやすかったり，彼女一途だったりと繊細ってことも発見しました。女ってのは繊細で優しくて誠実で，そんでもって力が強い男に弱いもんなのです。「ああん，男は性格と誠実さがいちばんだわねぇ～ん，セシルぅ，むっちゃ素敵。激Love。ローザ（セシルの恋人）嫌だけど許してあげよーじゃん」とほざくようになった私だって例外じゃああありません。この時点で彼の暗黒系がドーンとした容姿は気にならなくなりました。惚れてしまえば容姿なんざぁどうでもいいんです。男は心です。ね。

パラディンセシル

ところが，セシルはある時点で自分が暗黒騎士であることに限界を感じ始めました。確かに暗黒系の戦いは暗黒系モンスターに効きません。光と闇の属性があるこの世界だと，系統が同じ相手への攻撃はダメージが弱いのです。そしてモンスターってのはだいたいが暗黒系です。私もこれにはまいりました。そんなときセシルが暗黒騎士から光系の剣士であるパラディンにクラスチェンジできるチャンスに遭遇したのです。

しかし，クラスチェンジをすると，ようやく慣れ親しんだ彼の容姿が変わってしまうおそれがあります。サブウィンドウで拝見できる彼のお顔（ごつい兜姿）とか，フィールド上のちょこっとしたキャラクターだって絵が変わるでしょう。私はすっかりセシルのことが大好きになったので，このイベントはなんか嫌でした。しかし，弱いままだとこの先へ進めませんし，セシルご本人も暗黒の力を使わざるをえない自分に嫌気がさしている様子です。「セシルがどんな容姿になっても私の思いはきっと変わらないわ！」と自分を信じ，セシルをクラスチェンジができる試練の山へと向かわせたのです。

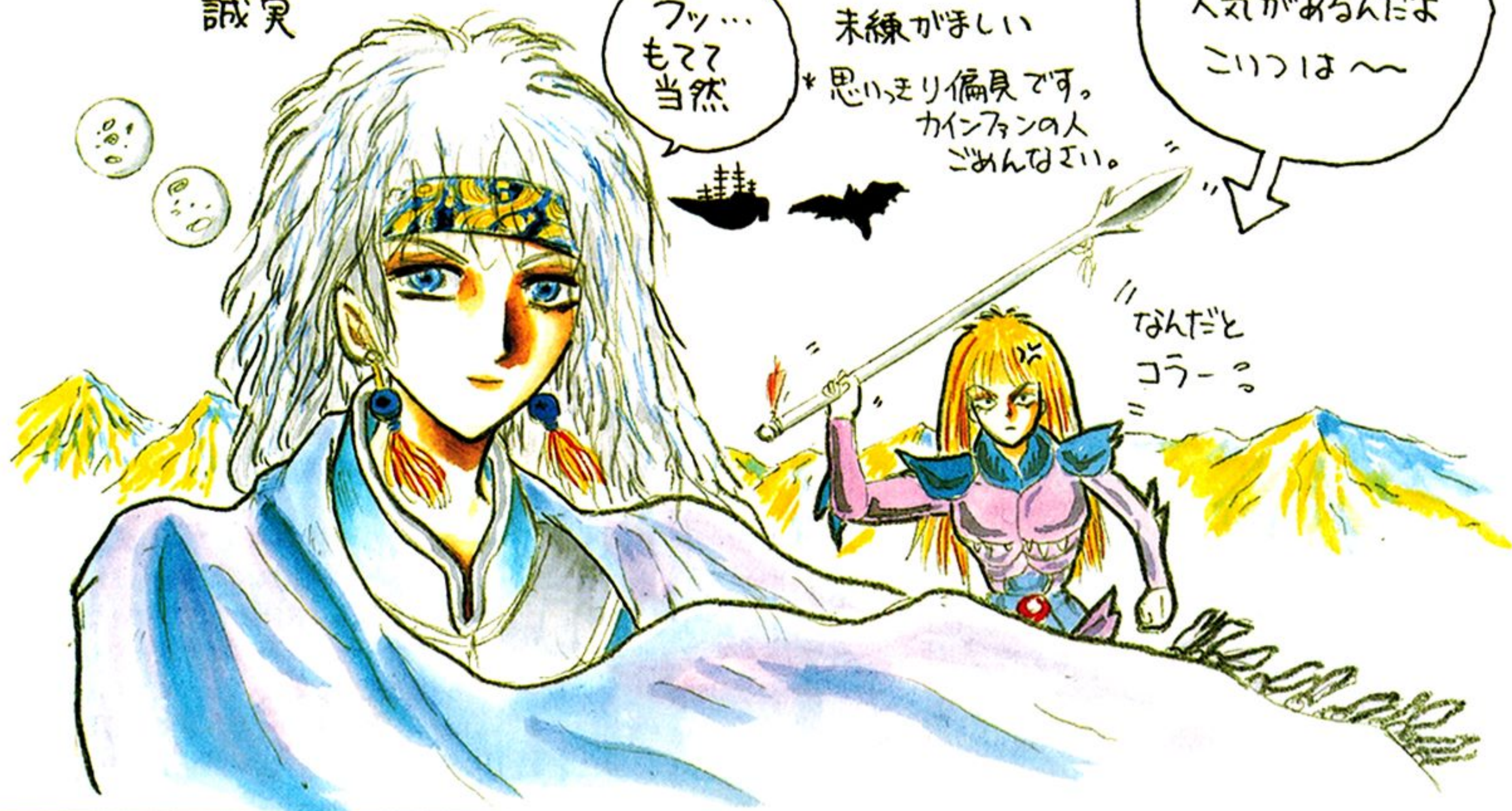
そして，セシルは自分の暗黒部分を倒し新しい光の力を得たのです。無事クラスチェンジをクリアです。えらいぞセシル。純粋に大喜びする私。彼がクラスチェンジをし終えると，フィールド上の彼のグラフィックが変化しました。「がーん，やっぱりお姿が変わっちゃったみたい」。予感は的中したのです。しかし，そのお姿はゲームの主役っぽい感じに変わっています。そのうえファンタジーにはお約束のマントまで羽織ってます。なんか以前より素敵っぽく見えます。

小さなドット絵ではわかりにくいので，私はそっとサブウィンドウを覗いてみました。なんとそこには，あのごつい兜をはずしたセシル様の『素顔』があるではないですか。その彼の素顔は切れ長の目に色素の薄い肌，髪は銀髪さらさらのいわゆる美形です。天野キャラに見られるミステリアスで耽美で華奢なお姿はファンタジーの主役の王道を突っ走った美しさです。

「なに，なに?! セシルってこんなに素敵なお顔だったのぉ！ すごい！ 美形じゃーん，よかったね～セシル!!」。そりゃあ「男は心よ～」ってのに間違いはありませんが，心に美しさが加わればうれしさは倍増ってもんです。このクラスチェンジのイベントはセシルLove Loveの私に完全なるトドメを刺しました。この時点でセシルは私にとって一生忘れられないゲームキャラクターになったのです。

★　★　★

パソコン通信やインターネットで文章のやり取りをして意気投合したペアは，出会ったあとに一気に結婚する場合があると話題になってます。文章で仲よくなるには，相手の心に訴えかける作業が必要です。心や性格で引かれた相手は容姿で引かれた相手以上に絆が深いものです。スクウェアも多少はこの効果を狙っていたとは思いますが，私は真っ向からこの演出にはまってしまったひとりでしょう。残念なことにFF Ⅳ以降，これ以上に主役に惚れ込むゲームには出合えてません。まあ，同じ演出をされても二度と同じような気持ちにはならないでしょうが。

初めてのスーパーファミコンのゲーム，新しいグラフィックやサウンド，性格から惚れ込んだ主役と，FF Ⅳは一生忘れられない作品となったのです。システムやグラフィックに凝ったゲームもいいですが，主役の魅力的な表現方法を考えたシナリオのゲームがあったらまたプレイしたいもんです。って，こんな惚れ方したのって私だけ？

Oh!X68000

X68000用新作ゲームの話

電脳倶楽部編集部
あいはらてつや　Aihara Tetsuya
aihara@mankai.co.jp

なんだって，いま頃X68000用のゲームソフトなんかと思われているでしょう，皆さん。ちなみに，最後のX68000用の市販ゲームはNEW（現在ではボクサーズロードとか，ちっぽけラルフの大冒険など，PSソフトを出しています）から出た，移植版のプリンセスメーカーでした，いままでは。当時，NEWの方には社長さんをはじめ，電脳倶楽部でも取材させていただいたことがありました。

あの頃はやっとインターネットが流行りだしたころで，プリンセスメーカーは完全に本数限定，インターネットオンリーの申し込み受け付けで，その数は，500本と聞いています。当時の印刷物の原稿などが資料として残っておらず（というのはなかなか痛いぞガイ＋ックス），グッズ，印刷物なども他機種版と同じものをつけたかったのだが，それができるぎりぎりの本数がそれだけだった，ということでした。実際にはあっという間に受け付け終了になってしまったわけですが，手にされた皆さんはラッキーなんですよ。さすがに，500本限定のソフトを開発期間をかけて出せば当然赤字，ということは重々承知のうえでの発売だったのでした。

と，いきなり脱線してしまいましたが，ここでひとつ教訓。出費を抑えるためには，いわゆる流通に乗せないで（おっと，こんなことソフトバンクの雑誌で書いていいのか？）「直販」すること，それから，たくさん作りすぎないこと，ということがここから引き出されます。

さて，いま，1000本のCD-ROMを作るとしたら（万の単位で作るともっと単価は安いけれど，そんなに作ってもしかたないですね），開発費を除いたもろもろの費用は200万円かかりません。100万円かからない，といいたいところですが，まあ，いろいろありますからね。それでもCDのマスタリングとデュプリ（プレス）代だけなら，1000本でも20万～30万円といったところでしょう。

もちろん，それに印刷物や箱，梱包発送費用，その他もろもろの部材の費用をかけても，200万円を超えない，ということなわけです。うまくやれば100万～120万円くらいに収めることも不可能ではないです。もちろん，開発費などは別になりますが。

パソコン用であれば，本体と開発キットその他一式がないといけませんし，CD-RドライブとCDライティングソフトがいるでしょう。CDレーベルや印刷物を作るには，原稿を作らなければならないわけで，Win or Macと，イラストレーターなどがないとできません。ですが，これらをすでに持っていれば，CD-ROMのマスターディスク（プレマスタ）も作れますし，もろもろの印刷物も完全版下で入稿までできてしまう世の中です。

標準的な本誌ユーザーなら，新たな出費をほとんどなしでCD-ROMを出すことも，いまや，そう手の届かない話ではなくなっているといえるかもしれません（もちろん貯えもいるが）。

ということで，今回満開製作所から出るゲームソフトも，もろもろの費用を考えて，ん～，なんとかなるかな？　という範囲でやっているんですね。初回1000本ですから，まあ，会社としてもうかるというにはほど遠い話ですが，面白いモノ，変なものを作って世に出す，というのが満開製作所のレゾンデートルです。

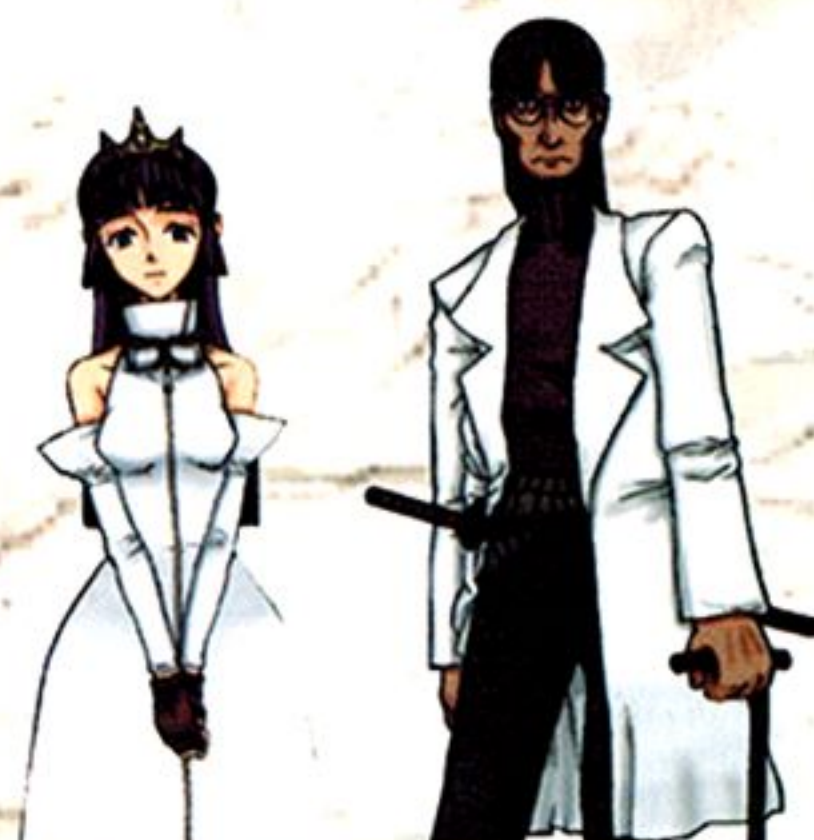
現在，このタイトル以外にも，もう1本別のソフトのリリースも決まっています。詳細は以下次号，ということになりますが，良質のゲーム作品をX68000で製品化することは，今後も取り組んでいきたいところなので，「実はフロッピーに収まらないようなゲームの現物を作ってしまったんですが……」という方はぜひ我々にも見せてください。ひょっとすると，現実に製品を出せるかもしれませんよ。まあ，移植ものというのは余計な出費がかかるうえ，権利関係が不透明なものも多いので無理かもしれませんが。

新作ゲーム裏話

さて，このノヴェルゲーム「あの，素晴らしいをもう一度」ですが，なんで，満開製作所からX68000用ゲームをこの時期出すことになったのか，多少ウラ話みたいなことをしておきましょうか。

これは満開の某プロジェクトに参加していた本作ディレクターのみかぜ氏があたためていた企画で，実は当初の某プロジェクトが××してしまったところから始まっています。なにかソフトを出さにゃならんよなあ，ということで，以前からの腹案として企画を出していた彼に白刃の矢が立った，というわけなのです。

もちろん，彼自身がやりたいというのでじゃあ，ゴーだ，ということになったわけです。

そんなこんながあるわけですが，企画についての会議の最初のやりとりで問題になったのは，まずタイトル。文字の間に空欄があるタイトルな

んて前代未聞です。ゲームはいうにおよばず，どんな種類の作品でもこんなにはひねくれていません。もちろん，商品として見た場合，広告や雑誌に載せるにしても，脱字とみなされかねず，相手先から嫌がられるのは必至です。普通の大人は絶対に避けて通ります。が，彼はガンとして受け付けません。まあ，満開製作所自体も普通じゃないですからね。でもって，これは空欄にしておくことに意味があるのだというモノなのですが，この空欄に込められたメッセージはどれだけのユーザーに届くのか，プレイの感想を楽しみに待ちたいと思います。

一事が万事このように考えてもらうとよいと思いますが，このゲームではディレクターの大胆な仕掛けが良くも悪くも特徴になるだろう，と思われます。システムの紹介のほうでは，アノスによって本の閲覧性と検索性を手に入れ……といったところがありますが，普通のゲームにあるようなバックログとは根本的に違うものです。あまり詳しく書くことはできないのですが，ビデオを再生しながら早送り，巻き戻しをすることを考えてください。こういう機能は普通は単純な頭出しにしか使えないわけで，その巻き戻しや早送りをしたところからは，ビデオならまったく同じ映像が再生されます。が，アノスでは，そこからやり直した場合，まったく同じものが見られるとはかぎらないのです，という風に考えてもらうのがいいかもしれません。

また，マルチエンディングとも一本道ともまったく違ったもの，というのがこのゲームのコンセプトのひとつになっています。この部分を膨らませて書くとあまりにもネタばれが多くなってしまうので，ここまでにしますが，普通は思いついても組むのが面倒でやらないとか，普通のプログラマが逃げ出してしまうようなからくりを持ったシステムであることは間違いありません。ただひとついえるのは，「みんなが当たり前だと思っていることのウラを見せる」というのが彼のクリエイターとしての身上です。

さて，もうひとつ。このゲームの音楽は電脳倶楽部のオリジナル曲ではお馴染みの水野裕之氏が担当しています。BGMはもちろん，歌の作曲，アレンジまでヤマハのMU80ひとつでこなす彼ですが，すべての曲はCD-DAおよびFM音源で収録されます。ぜひここは，CD-ROMから直接再生しながらプレイしてもらいたいところです。CD-ROMはオーディオ出力につなぎ，効果音はX68000本体からという組み合わせになりますが，ぜひ，ベストな環境で聞いていただきたいと思うところです。これもX68000でCDトラックを標準で聞いてゲームができるという点では初の商業作品になります。

なんだか奥歯にモノが詰まった話になってしまいますが，9月9日発売です。ぜひご期待ください。

column

あの、素晴らしい　　をもう一度

かざみみかぜ。
mikaze@mankai.co.jp

プロローグ

目覚めたとき、主人公には過去がなかった。『記憶喪失』。物語で使い古されたような記憶障害だろうと、自らがその状況に置かれたのなら話は別。混乱と茫然とめまいがひとまとめになって、結果的に出てくるのは乾いた笑い。右を見ても左を見ても見渡す限りの荒原の中で絶望と放心にその身を委ねようとしたところ、ふいに見つけたのは自分と同じように倒れていたひとりの少女。

わずかな希望にしがみつくように彼女を抱き起こすと、少女は自分のことを知っていた。だが同時に彼女の時間は、相当に切羽詰まった出来事の続きにあるようだった。わめき立てながら何度も何度も必死に「あれからどうなったの？！」という言葉を連発する。

「それはこっちが聞きたいくらいだ」という言葉を抑え込み、混乱の海に落ちてしまいそうな自らの気持ちを必死に制御しながら『記憶がない』ことを説明すると、彼女は世界の終末がきたような顔をする。

「それはこっちがしたい顔だ」という気持ちを抑え込んでいると、今度は自分の胸に飛び込んでもの凄い勢いで泣き出した。

錯乱寸前の気持ちは、傍らの少女の混乱をなだめることによりいつか霧散し、まずはこの荒原を脱するために歩き出すことになる。

そして歩き回った果てにようやく見つけたのは小さな村。ともかく疲弊しきった体を休息させるのが先決とその日は村の宿屋で床につく。

だが明けた次の日、主人公は隣の部屋で寝ていたはずの彼女の悲鳴で目を覚ます。驚いてそこに行ってみると彼女は自分を見つけた瞬間、叫び出した。

「あれからどうなったの！？」

と。

彼女の言動はしばらく止まらず、『記憶の頼みの綱』である彼女の錯乱に困惑するばかり。

しかし、しばらく彼女の言動を聞いているうち、ひとつの答えが見えてくる。

見えた理由は『彼女は昨日目覚めたときと同じ言動を繰り返している』から。

そして導き出された結論は『彼女は昨日１日の記憶が完全に欠落している』。

村の病院に連れていくと、医者が出した答えは驚愕の病名だった。

『繰り返し病』

普通、記憶喪失は過去の記憶がなくなるが，繰り返し病は『ある日』を基準としたその先の記憶に対し、不定期に『リセット』がかけられる。そしてその結果『ある日』を何度も何度も繰り返してしまう。

それは何度も何度も永遠に記憶の蓄積をやり直し、未来を手に入れることが許されない悲運の輪廻。

主人公の口からふいに出てきたのはひとつの言葉。

「人は間違ったらやり直すべきだ。でも繰り返すべきじゃない。そんなことが本人の意志と関係なく行われるなんて許されていいはずがない」

滑らかに出てきたその科白は、もしかしたら過去の自分がいい慣れた言葉なのかもしれない。

そして２人は旅に出る。

互いに欠けた『過去』と『未来』を手に入れるために。

もし互いに欠けたものがあるとしても、それを２人で補いあうことができれば補って余りある以上のものが手に入るはずだから。

「物語」で「ゲーム」するために

『あの、素晴らしい　　をもう一度』は文章に絵と音が加わった状態で物語が進行していくノベルタイプのゲームです。

ただしこのソフトでは「物語」で「ゲーム」することを目的のひとつにしています。

物語を重視するあまり、ゲーム性をおろそかにすることも、ゲーム性を重視するあまり物語をおろそかにすることもないゲームデザインをしようということです。

このため、物語や設定を最大限に活かすゲームシステムを考案し、そのシステムを最大限に活かす物語や設定を組んでいくというアプローチをとっています。

そして卵が先か鶏が先か、結果的にどちらかわからなくなるほど「シナリオ」と「システム」が融合されたとき、目的は達成されると考えています。

そのために生み出されたのが、Advanced Novel Operation System。通称ANOS（アノス）です。これは複数のセーブデータによるセーブ＆ロードを繰り返さなくとも、一度見たシナリオであれば自由に物語を辿り直し、選択肢を選び直すことができるシステムです。これによりノベルゲームは本の検索性と閲覧性を手に入れ、ノベルゲームをアドバンスドノベルへと進化させます。

ゲーム中，プレイヤーは物語の節々でさまざまな選択をしていくことになりますが，どれかひとつを選択しているだけではベストエンドを迎えることはできません。それは『もうひとつの選択』の行く末を知っていないとわからない謎が多々存在するからです。

そのため，プレイヤーは物語の中でAngel Omnipotent Stone、通称ANOS（アノス）と呼ばれる天使のひとかけらを手に入れます。それによりいままでに進めた物語を自由自在に駆け巡り，選ばなかった選択肢を選び直すことができるようになります。そして選択した両方の結果を目にして初めてわかる真実をもとに、世界の謎を解き明かしていくのです。

内蔵音源を駆使した高品位ステレオPCM再生

鎌田 誠 Kamada Makoto

はじめに

X680x0の内蔵音源は，8チャンネルのFM音源(OPM：YM2151)と，15.625kHzまでのサンプリング周波数で再生できる1チャンネルのPCM音源(AD PCM：MSM6258V)で構成されています。PCMデータをステレオ再生する(左と右で別々のPCMデータを再生する)ためには最低でも2チャンネルのPCM音源が必要ですから，ハードウェアを追加しない限り，普通の方法ではPCMデータをステレオで再生することはできません。しかし，限られた内蔵音源を駆使してPCMデータをステレオ再生する試みは過去にいくつかありました。

X680x0で広く公開されたPCMのステレオ再生プログラムは，古くはOh!X 1993年3月号の〝(で)のショートプロぱーてぃ〟のコーナーに掲載された秋山嗣晴氏によるPCMST.Sに遡ります。PCMST.Sによるステレオ再生の方法は，ステレオのPCMデータを細切れにしてAD PCMで左右交互に出力し，左右の切り換えを高速に行うことでAD PCMを疑似的にステレオ化するというものでした。私の知る限り，X68000の内蔵音源のみでPCMデータをステレオ再生する試みを広く発表されたプログラムとしてはおそらく初めてであり，私も当時その発想と実現力に感動を覚えつつOh!Xに掲載されていたプログラムをせっせと入力した覚えがあります。

ただ，PCMST.Sの方法では実際に左右同時に(意図した)音が出るわけではなく，また，PCMデータを細切れにしているためにノイズが多いのが難点でした。なお，このプログラムはその後バージョンアップされて電脳倶楽部VOL.99(1996年8月号)にPCMST.XとWAVST.Xとして掲載されています。

電脳倶楽部にPCMST.Xが掲載されたことがきっかけで，伊倉〝DigitalJunkie〟晴読氏が別のアプローチでPCMデータをステレオ再生するプログラムSTPLAY.Xを作り，電脳倶楽部VOL.100(1996年9月号)に掲載されました。STPLAY.Xは，右側の音をAD PCMで，左側の音をFM音源で再生するという方法で，左右同時に15.625kHzでPCMを再生することができました。

FM音源を4チャンネル使うことで1チャンネルのPCM音源をエミュレートできることは同氏の手によって電脳倶楽部VOL.78(1994年11月号)に掲載された〝志村けんX〟というプログラムですでに証明されており，STPLAY.Xはそれを応用したものでした。

今回の試み

さて，私が激光電脳倶楽部VOL.6に向けて音に関係する記事を書くことになったとき，私は自分もPCMのステレオ再生を試みようと思いました。それも激光電脳倶楽部なのですから，CD2PCMT.Xなどを使うことで音楽CDから容易にデータを供給できる(もちろん，個人的な使用の範囲でですが)サンプリング周波数が44.1kHzのステレオのPCMデータを直接再生できるプログラムを作りたくなりました。

前述のとおり，X680x0の内蔵AD PCM音源は最高で15.625kHzまでのサンプリング周波数にしか対応しておらず，AD PCMでこれよりも高い周波数でPCMデータ再生することは不可能です。しかし，STPLAY.Xで使われていたFM音源の処理能力にはまだ少し余裕があります。そこで，私は15.625kHzのAD PCMの使用はさっさと諦めて，左右両方のチャンネルをFM音源で再生して再生周波数をAD PCMよりも上げられないかと考えました。

FM音源の処理速度の限界

諦めついでにもうひとつ。アセンブラでFM音源レジスタを直接叩いたことがある人ならば知っていることですが，FM音源レジスタの読み書きはFM音源アドレスポート($00E90001)とFM音源データポート($00E90003)という2つのポートをアクセスすることによって行います。ここで，それぞれのポートに書き込んだあと，FM音源データポートの書き込みビジーフラグ($00E90003のMSB)がクリアされるまで次の書き込みを行ってはいけないという決まりがあります。FM音源データポートへの書き込みのあとビジーフラグがクリアされるまでにかかる時間は，FM音源LSI(YM2151)の入力クロックの68クロック分です。

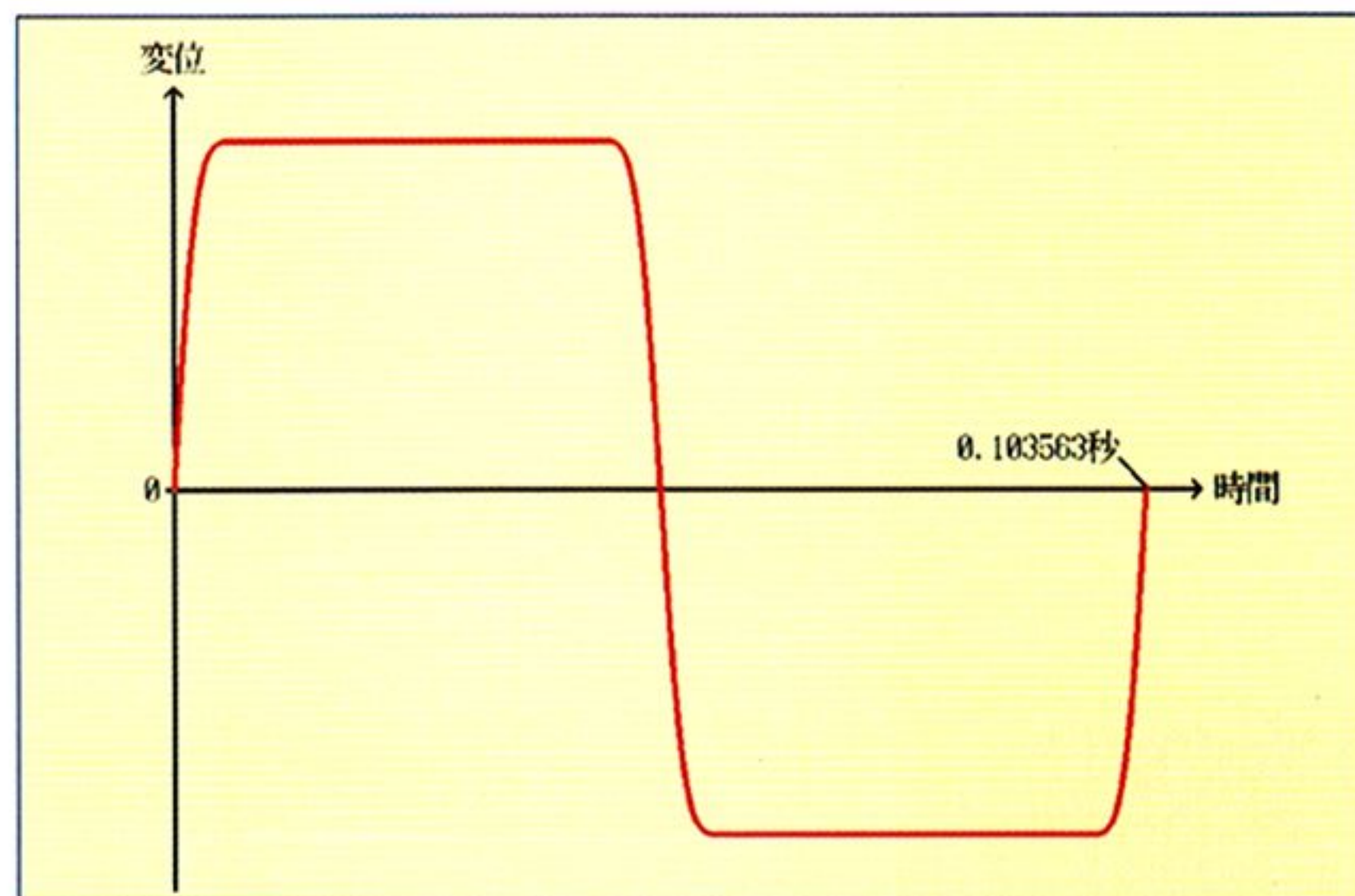



図1　矩形波に近い音の波形

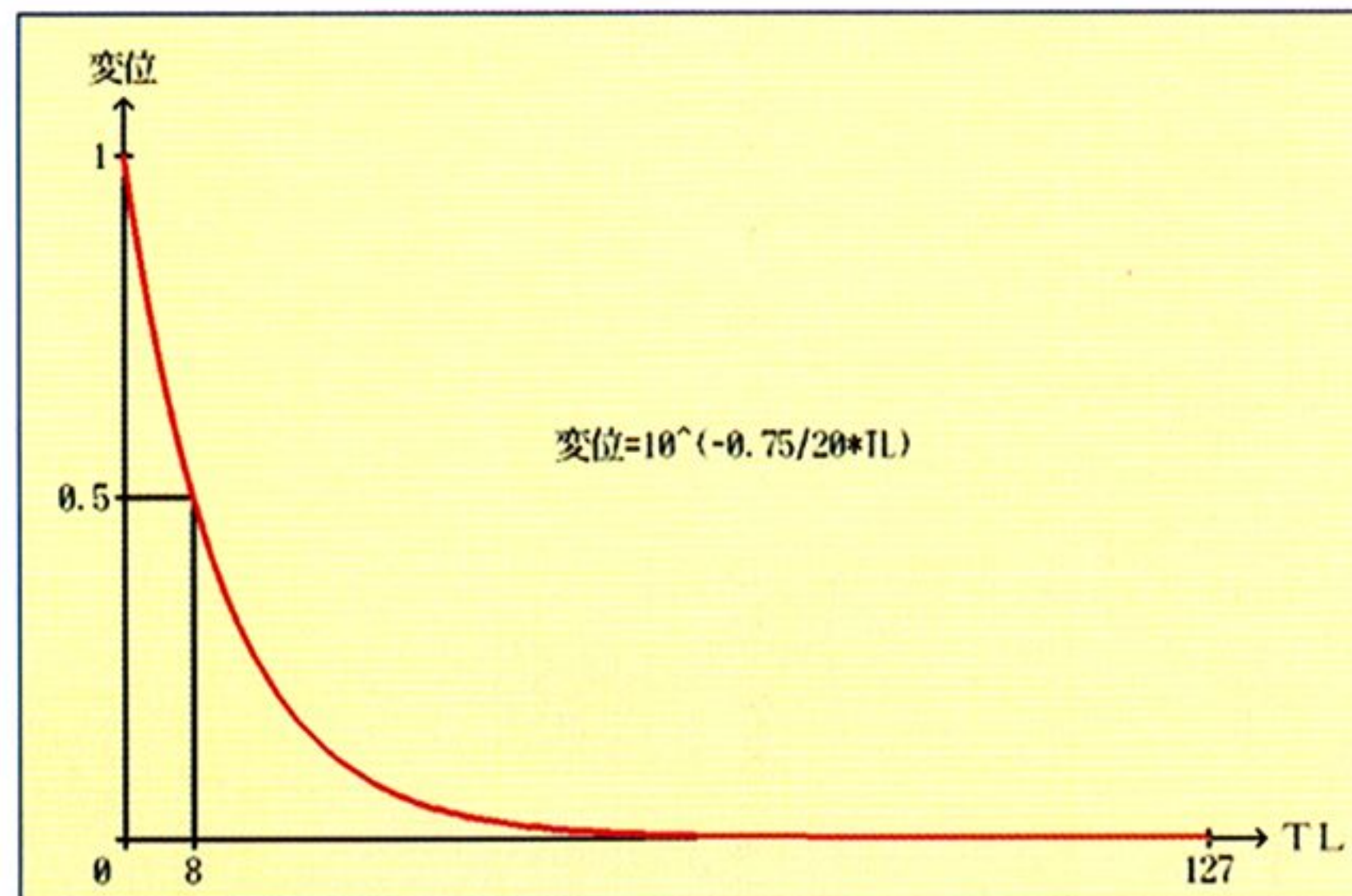



図2　TLと変位の関係

YM2151には4MHzのクロックが入っていますから，FM音源データポートへの書き込みを繰り返す場合には少なくとも17 μ秒以上間隔をあけて行わなければならないことになります。実際にこれよりも短い間隔でFM音源データポートに書き込もうとすると，書き込みに失敗して期待どおりの音が出なくなります。

音楽CDのサンプリング周波数は44.1kHzです。単純に割り算をすると，1周期あたり約22.6757 μ秒しかありません。しかもステレオですから，1周期の間に2チャンネル分のデータを処理しなければなりません。

FM音源データポートへの書き込みは1回あたり最低でも17 μ秒以上かかり，2チャンネル更新するためにはTL(トータルレベル：要するに音量制御部)を2つ更新するため少なくとも2回データポートに書き込まなければなりませんから，17×2＝34＞22.6757であり，実際に動かして確かめるまでもなく，FM音源のみでPCMデータをステレオで44.1kHzという周波数で再生することが不可能であることがわかります。

しかたがないので，44.1kHzでのステレオ再生は諦めて，周波数を落としてステレオ再生することを考えましょう。

FM音源でPCMデータを再生する方法

ここで説明する方法は伊倉"DigitalJunkie"晴読氏が"志村けんX"やSTPLAY.Xで使用した方法を私が改良したものです。

・矩形波に近い音を作る

まず，FM音源を1チャンネル使って，波長が非常に長くてなるべく矩形波に近い音を作ります(図1)。この波形が，変位が最大になった状態をどれだけ安定して維持できるかが，再生できるPCMの音質に影響してきます。

矩形波に近い音のパラメータ(FM音源レジスタに設定する値)は，次のような方法で独自に決定しました。

FM音源のレジスタにどのようなパラメータを指定するとどのような音(波形)が作られるのかを数値計算で予測してグラフを描き，それがなるべく矩形波に近づくようにパラメータを調整して，目的にあったパラメータを見つけるのです。

変調を強くかけたほうが矩形波に近づけやすいので，ここではコネクションを0にして4つのスロットを直列に接続します。フィードバックは使いません。すると，生成される波形は時刻tに関する次のような関数で予測できます(波形を予測するための式は『Inside X68000』の記述を参考にしました)。

```
変位(t)=α(C2)*sin(MUL(C2)*t
        +α(M2)*sin(MUL(M2)*t
        +α(C1)*sin(MUL(C1)*t
        +α(M1)*sin(MUL(M1)*t))))
 α(M1)=10^(-0.75/20*TL(M1)+e/2)
 α(C1)=10^(-0.75/20*TL(C1)+e/2)
 α(M2)=10^(-0.75/20*TL(M2)+e/2)
 α(C2)=10^(-0.75/20*TL(C2)+e/2)
MUL(n)=スロットnのMUL(周波数の倍率)で
         掛ける値(0.5,1～15)
TL(n)=スロットnのトータルレベル(0～127)
t=時刻
e=2.718281828459(自然対数の底)
```

グラフを描きながらMUL(n)とTL(n)を少しずつ変化させ，なるべく波長が長くて矩形波に近くて最大変位が安定している波形を生成するパラメータを探した結果，次のようなパラメータが得られました。

```
  MUL(M1)=1,MUL(C1)=1,MUL(M2) =1,MUL(C2)=0.5
  TL(M1)=45,TL(C1)=36,TL(M2)=34,TL(C2)=0
または，
  TL(M1)=47,TL(C1)=38,TL(M2)=35,TL(C2)=0
```

この矩形波に近い音の波長は，ほぼ0.103563秒であることが実験的にわかっています。

・変位の制御

コネクション0で生成される波の振幅は，キャリア2(C2)のトータルレベル(TL)を変化させることで，キーオンしたままリアルタイムに変動させることができます。TL=0のとき振幅は最大になり，TL＝127で最小(無音)になります。矩形波が最大変位を安定して維持している期間は，振幅が変位そのものを表しているので，C2のTLを操作することでリアルタイムに変位を指定できることになります。TLを変化させることで生じる矩形波の変位の変動が，FM音源でPCMデータを再生する場合の音の素になります。

なお，TLで指定した数値に対して変位が指数関数的に変化するのに対して，一般的にPCMデータは直線的な値なので，PCMデータからTLに与える値を求めるためには対数関数による変換が必要です。具体的には，TLに応じて出力される変位は，

```
L=10^(-0.75/20*TL)
```

という式で計算されるLに比例します(図2および図3)。

PCMデータは，大雑把ないい方をすれば音の波を一定の間隔で細切れにして得られる変位を並べたものですから，それを対数関数で変換してC2のTLに流し込んでやれば，FM音源でPCMデータを再生できるということになります。なお，1データごとに対数関数を計算していたのでは遅くてしかたがないので，実際にはPCMデータの変位からTLへの変換にテーブルを使います。16ビットのPCMデータを7ビットのTLに変換するために64KBのテーブルを使います。

・複数の矩形波を重ねる

さて，TLで変位を自由に操れるといっても，矩形波(に近い音)がひとつだけでは安定して使える期間が短すぎて，PCMデータを長時間連続して再生することができません。そこで，この矩形波に近い音をCh1～Ch4の4つのチャンネルでCh1→Ch2→Ch3→Ch4→Ch1→……の順序で，なるべく正確に1/4周期ずつずらして発生させます(図4)。

このあと，Ch1～Ch4がそれぞれ安定して最大変位を維持している範囲を1/4周期ずつピックアップし，それぞれその範囲でTLを操作します。

・STPLAY.Xの場合

STPLAY.Xでは，PCMデータをTLに変換して乗せるチャンネルの順序が常にCh1→Ch2→

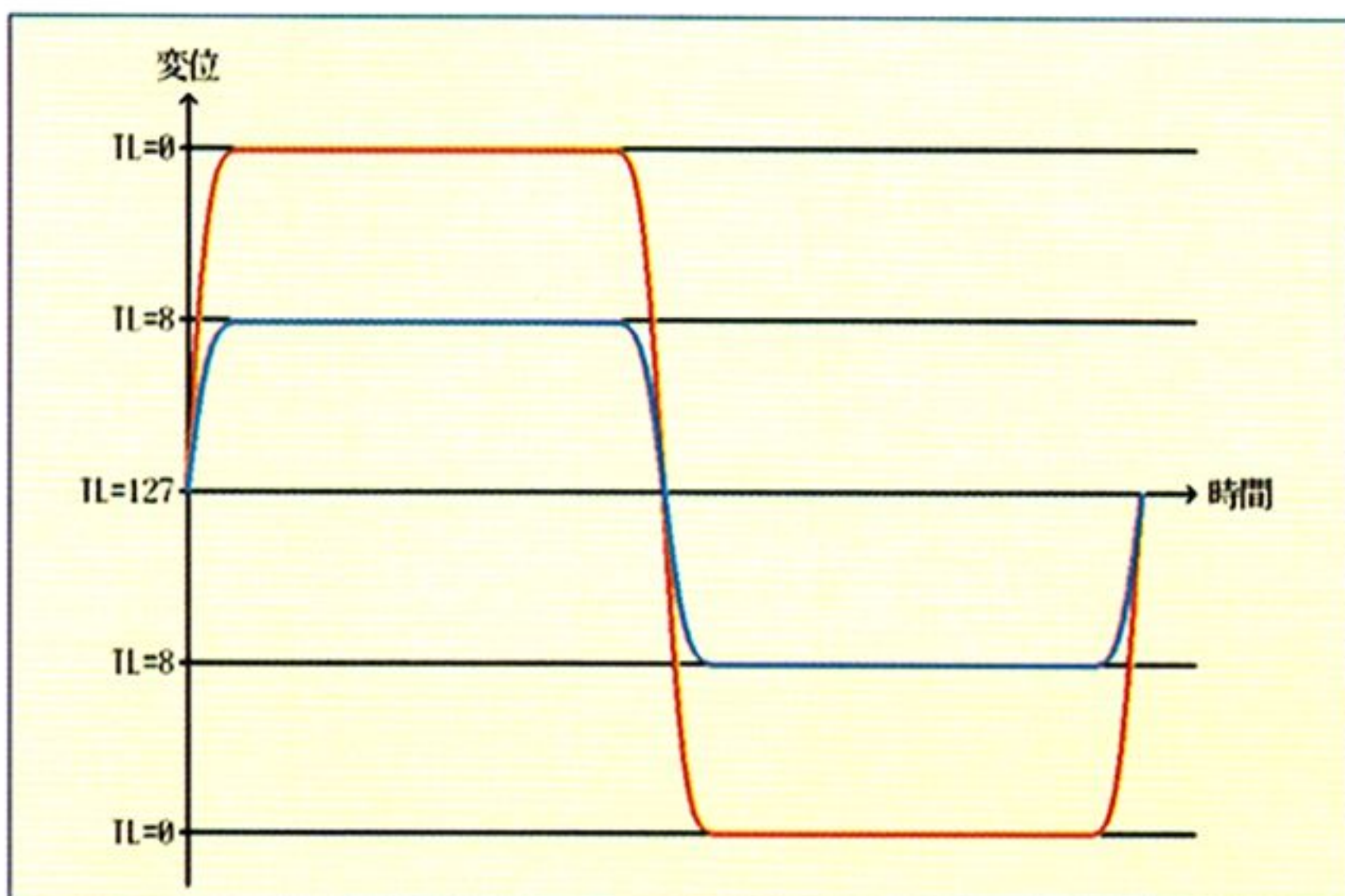



図3　矩形波に近い音の最大変位をTLで動かす

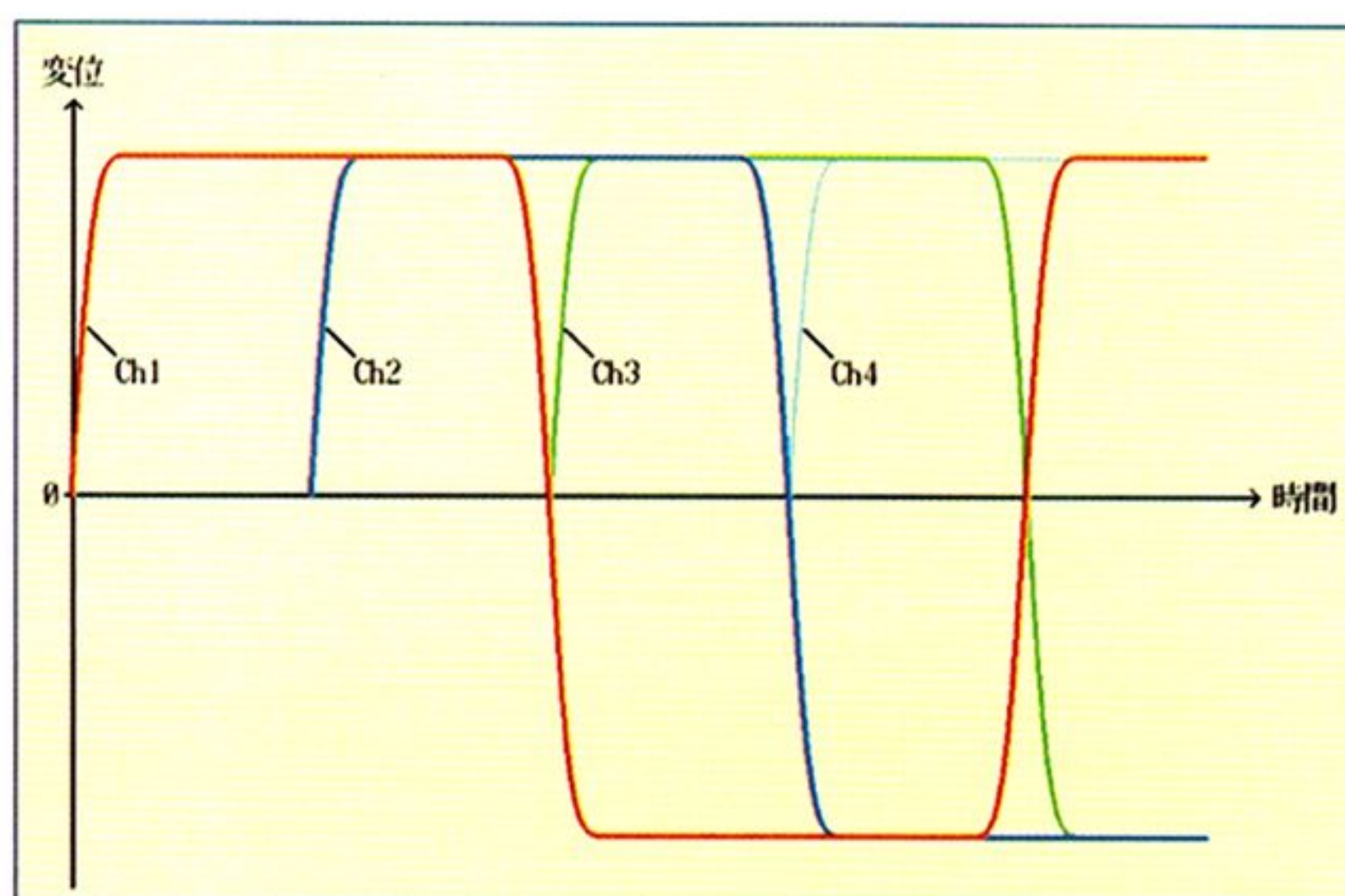



図4　矩形波に近い音を1/4周期ずつずらしてキーオンする

Ch3→Ch4 →Ch1→……となっていて，使用していないチャンネルはすべて最大変位(TL=0)の半分の変位(TL=8)に固定されていました。この方法では，Ch1とCh3，Ch2とCh4がそれぞれ1/2周期ずつずれていますから，使用するチャンネルの変位をTL=8の前後で高速に変動させてやれば，最終的には変位0を挟んで上下に高速に変動する波が作り出されます(図5)。

この方法の欠点は，出力される音の最小の変位をTL＝8に割り当ててあり，図2からわかるようにTL＝8の近辺では変位の分解能が低いために波形の再現性が悪く，音量に関係なく大量の高周波ノイズが発生してしまうことです。

・新しい方法

私がとった方法は，見かけはSTPLAY.Xの方法よりもむしろ簡単です。Ch1とCh3，Ch2とCh4はそれぞれ1/2周期ずつずれているので，Ch1の変位がプラス側にあるときCh3の変位はマイナス側にあります。そこで，Ch1の変位がプラス側にある期間は，＋側のPCMデータをCh1に，マイナス側のPCMデータをCh3に乗せるのです。使っていないチャンネルはすべて無音(TL＝127)に固定します。ほかのチャンネルを使うときも同様です(図6)。

この新しい方法では，出力される音の最小の変位をTL＝127に割り当ててあり，図2からわかるようにTL＝127の近辺では変位の分解能が非常に高いので，波形の再現性がSTPLAY.Xの方法よりも圧倒的によくなります。

・変位の符号に応じたチャンネルの切り換え

ところで，当然のことですが，変位がプラス側にあるときとマイナス側にあるときで使用するチャンネルを変えるということは，変位の符号が変わるたびに使用するチャンネルを切り換えなければならないということです。チャンネルを切り換えるにはそれまで使っていたチャンネルを無音にしたうえで，次に使うチャンネルにデータを乗せなければなりません。すると，チャンネルを切り換える瞬間に，最短でもFM音源レジスタをひとつ更新する(ひとつのチャンネルを無音にする)のにかかる時間だけ，本来のデータと異なる変位を出力してしまうことになります。たとえ一瞬でも局所的に本来のデータと大きく異なる変位を出力するとプチノイズになってしまうので，いかにしてなるべく正しい変位を出力しつつチャンネルを切り換えるかが重要になってきます。

変位の符号が変わるということは，極端な高周波成分を含んでいない限り，その前後の変位は0に近いはずです。そこで，0を挟んで変位の符号が変わっているところの前後(どちらか一方がプラス側で他方がマイナス側)で，前後のうち変位が0に近い側の変位を強制的に0にすることにします。波形が少し乱れますが，前述のように極端な高周波成分が含まれていなければもともと0に近かったデータですから，音質の大きな劣化はないはずです(図7)。

変位が0になっていれば，チャンネルの切り換えは簡単です。変位が0のデータを出力すればそのチャンネルは無音になりますから，そのまま次のデータから次のチャンネルに移ることができます。

1/4周期ごとのチャンネル遷移

ところで，Ch1～Ch4がそれぞれ安定して最大変位を維持している範囲を1/4周期ずつピックアップすると書きましたが，実はこれが非常にシビアな処理になります。再生が続いている間は4チャンネルともキーオンしたままですから，波の位相はMPUの動作とは無関係に正確に進行していきます。これに追従して正確なタイミングで使用するチャンネル(TLを操作するチャンネル)を切り換えないと，あっという間に最大変位が安定している範囲を外れ，期待どおりの音が出なくなってしまうのです。

STPLAY.Xでは上記の音の波長の値が不正確で，短時間で破綻してしまうため数分間隔でキーオンからやり直すことになっており，キーオンし直すときに異音が出てしまっていました。今回私は実験で上記の音のかなり正確な波長として0.103563秒という値を得，少なくとも1時間程度ならばキーオンし続けても破綻しないようにすることに成功しました。

実際に矩形波に近い音の波長を管理する方法は以下のとおりです。

まず，非常に短い間隔でFM音源レジスタを更新するために，TIMER-Dによる割り込みを使います。TIMER-DはCONFIG.SYSにPROCESS＝の指定をしたときにHuman68kがバックグラウンド処理のために使用しますが，PCMを再生している最中にバックグラウンドタスクを回すだけのMPUパワーが残っているとはとても思えませんから，ためらわずにTIMER-D割り込みをHuman68kから奪って使います。

TIMER-D割り込みの割り込み間隔は1μ秒単位で指定できますが，FM音源レジスタをひとつ更新するのに18μ秒以上かかるので，割り込み間隔をあまり短くしても意味がありません。そこで，X68030の場合はデフォルトで22μ秒間隔で割り込むことにします。周波数に換算すると45.455kHzになります。ステレオ再生時は1サンプリングあたりFM音源レジスタを2回更新する必要があるので，22727kHzになります。これは44.1kHzの半分の22.05kHzよりも少し高い周波数です。

FM音源でPCMを再生する場合，もっとも厄介なのが，FM音源で発生させた矩形波に近い音の波長の管理です。前述のとおり，FM音源で発生している音の波長は0.103563秒であり，割り込み間隔の22μ秒で割ると1周期あたりの割り込み回数が約4707.41回ということになります。

キーオンは4707.41/4＝1176.85より割り込み回数にして1176～1177回の間隔で行えばよいとして，そのあとの1/4周期ごとのチャンネル遷移をいかに正確に行うかが問題です。22μ秒という短い間隔の割り込み要求をMPUが1回も取りこぼさないという保証はなく，1周期あたり1回ずつずれただけでも誤差が蓄積して数秒から数十秒で破綻してしまいます。特にあとでディスクアクセスをしながらPCMをステレオ再生しようと

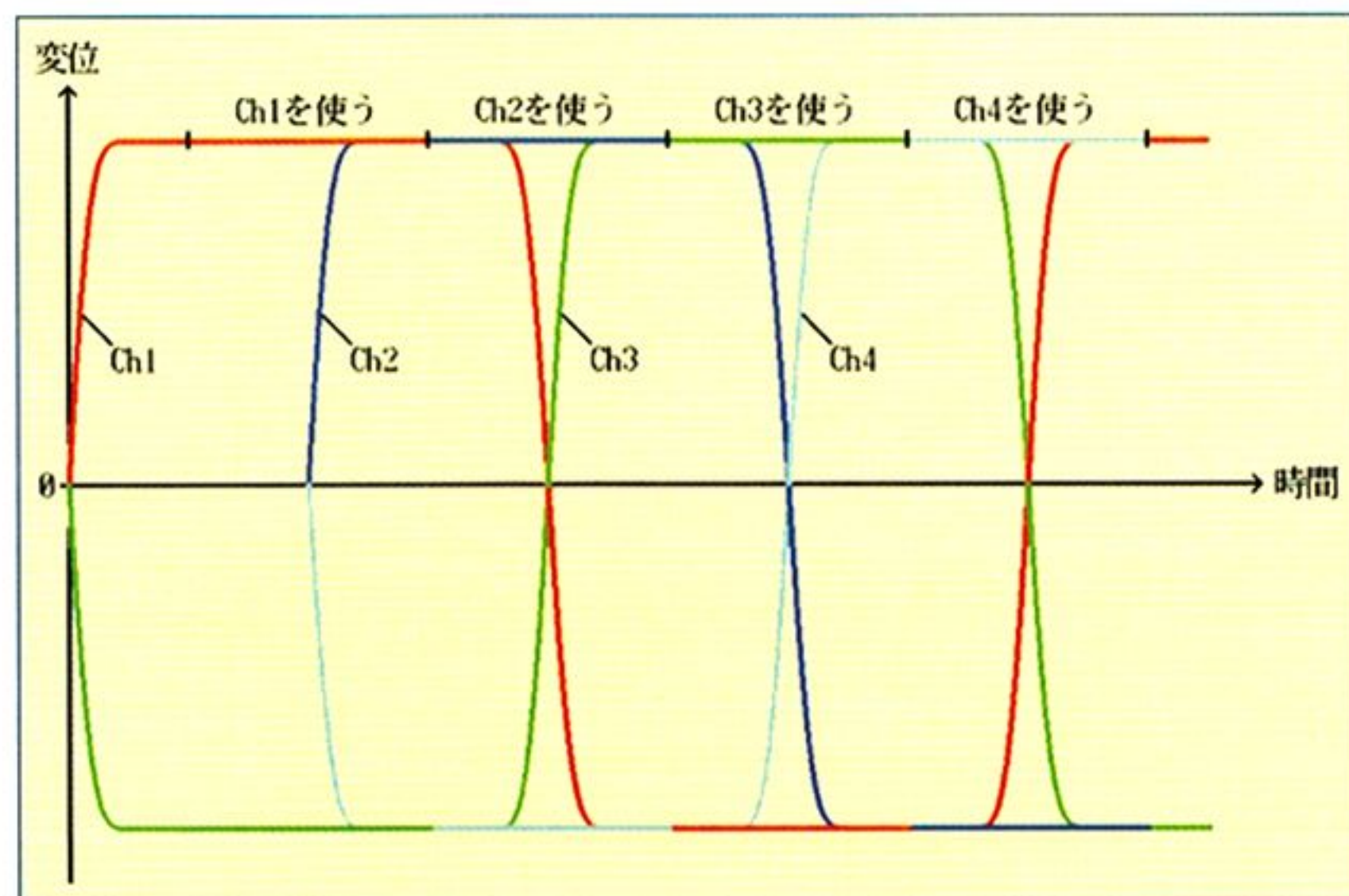



図5　矩形波に近い音の＋側だけ使う

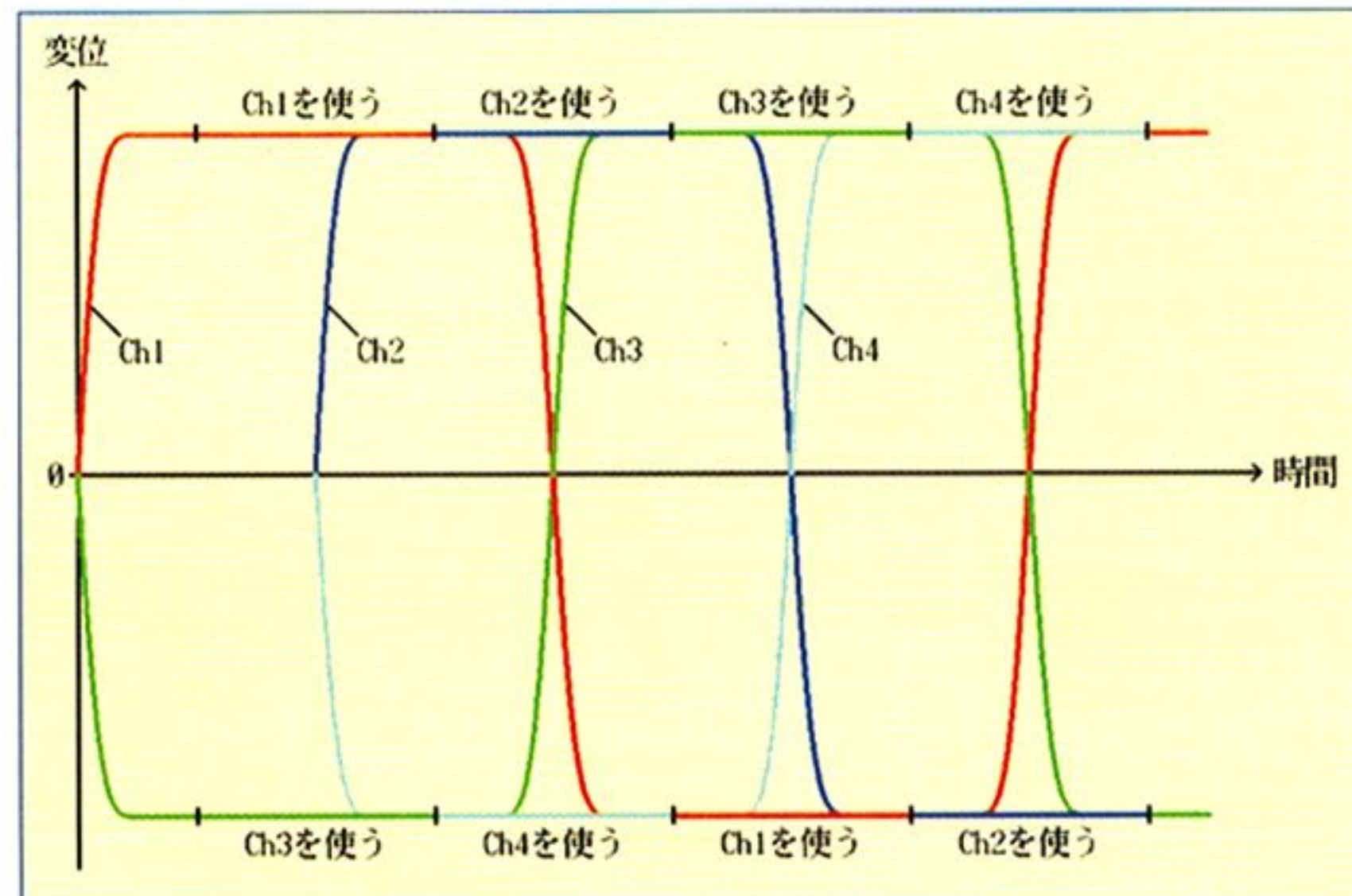



図6　＋側と－側で違うチャンネルを使う

していることを考えると，割り込みを取りこぼすのは必至で，TIMER-Dの割り込み回数だけに頼っていたのでは波長の管理はできないという結論に達します。

そこで，TIMER-Cのカウンタを併用することにします。TIMER-Cは普段は1/100秒カウンタとしてカーソルの点滅などに使用されているものです。TIMER-Cを16μ秒単位でダウンカウントするモードにして，256回でオーバーフローするように初期値を与えます。すると0.103563/(0.000016*256)=25.283935546875，0.283935546875*256=72.6875となって，1周期の間にTIMER-Cは正確に25回オーバーフローしてから元の値よりも72.6875だけ減ることがわかります。これを利用して，TIMER-DのカウントのみでTIMER-Cが25回オーバーフローする頃まで進んでから，残りの時間をTIMER-Cが前回よりも72.6875小さい値になるまでTIMER-D割り込みの中でTIMER-Cのカウンタを監視しながら進めることで，1周期の終わりを決定します。実際にはTIMER-Cのカウンタは整数なので，72.6875×256＝18608から「ワードサイズのワークから毎回18608ずつ引き，その上位バイトを採用する」という方法でTIMER-Cの値を予測して比較します。

この方法によって0.103563秒という微妙な波長を(平均して)きわめて正確に維持することが可能になり，1時間くらいキーオンしたままでも大きな誤差が生じず，再生し続けることができるのです。

ところで，1/4周期進んだところで強制的に次のチャンネルに切り換えてしまうと，変位が大きな区間で切り換えることになった場合にプチノイズが発生してしまいます。今回採用した矩形波に近い音は，独自に作り出したパラメータで生成されていることもあって，最大変位が安定している範囲が1/4周期の0.02589秒よりも約70%長い0.04423秒あることがわかっています。つまり，あまり厳密に1/4周期ごとにチャンネルを切り換えなくても，チャンネルを切り換えるタイミングが多少前後した程度ならば問題なく再生できるのです(もちろん，長時間再生したとき1周期の長さの平均が0.103563秒になるように厳密に制御する必要があります)。

変位の符号が変わるときに変位を強制的に0にしているので，1/4周期が経過しても少しの間はチャンネルを変えずに変位の符号が変わるのを待ち，変位の符号が変わった瞬間に符号と同時に1/4周期ごとのチャンネルの遷移も行うことにしました。このようなチャンネル遷移の遅延はあまり長くはできませんが，たいていのPCMデータは符号が頻繁に入れ替わるので，少し待つだけでプチノイズはほとんど除去されます。大きな低周波成分があるときは少し待っても変位の符号が変わらないことがあるので，そのような場合は途中で諦めて強制的にチャンネルを遷移します。

ステレオ化

前述のように，FM音源でPCMを再生するためには4チャンネル必要です。FM音源には全部で8つのチャンネルがありますから，すべてのチャンネルを使えばFM音源のみでPCMをステレオ再生することができます。

注意しなければならないのは，FM音源レジスタは2つ同時に更新できないということです。そのため，右側の変位を更新するタイミングと左側の変位を更新するタイミングが微妙にずれてしまいます。この問題を解決するには，PCMデータの周波数を変換するときに，左右の出力データのサンプリング位置をサンプリング間隔の半分だけずらしておかなければなりません。

STPLAY.XではFM音源が担当する音が片側だけだったので，たとえばCh1を使っている間はFM音源アドレスポートをCh1のTL($78)に固定してFM音源データポートだけを更新し続けることができました。しかし，ステレオ化する場合，Ch1のTL($78)とCh5のTL($7C)を交互に更新するためにFM音源アドレスポートも毎回更新しなければなりません。そのため，FM音源によるPCMのステレオ再生の負荷はモノラル再生の2倍以上になります。

サンプリング周波数の変換

入力データのサンプリング周波数とFM音源でPCMを再生する際の出力時のサンプリング周波数は異なる場合が多いので，PCMデータのサンプリング周波数の変換が必要です。サンプリング周波数の変換は，PCMデータをOPMデータに変換する前に行います。すなわち，PCM→PCMのサンプリング周波数変換を行います。

S44PLAY.Xでは，次の3通りのサンプリング変換方法を選べるようにしてみました。

(1)間引き・伸張
(2)直線補間
(3)面積補間

(1)間引き・伸張

サンプリング間隔が短くなる場合は，データの伸張を行います。1データごとにサンプリングのタイミングの誤差を蓄積し，誤差が1データ分に達したら最後のデータを重複して出力します。

サンプリング間隔が長くなる場合は，余った入力データを捨てます。

この方法は非常に簡単なのでリアルタイムに変換できますが，音質はよくありません。

(2)直線補間

入力されたPCMデータを直線で補間したうえで，出力側のサンプリング間隔でサンプリングし直します(図8)。

この方法では1データあたり1回の乗算が必要なので，68000ではリアルタイムに変換できません。

(3)面積補間

面積補間という呼び方は筆者が勝手につけたものです。

入力されたPCMデータを直線で補間したうえで，出力側のサンプリング間隔で刻み，切り取られた各区間の平均の高さ(面積に比例する)を出力データとします(図9)。

この方法では1データあたり2回以上の乗算が必要なので，リアルタイム変換ができるのは事実

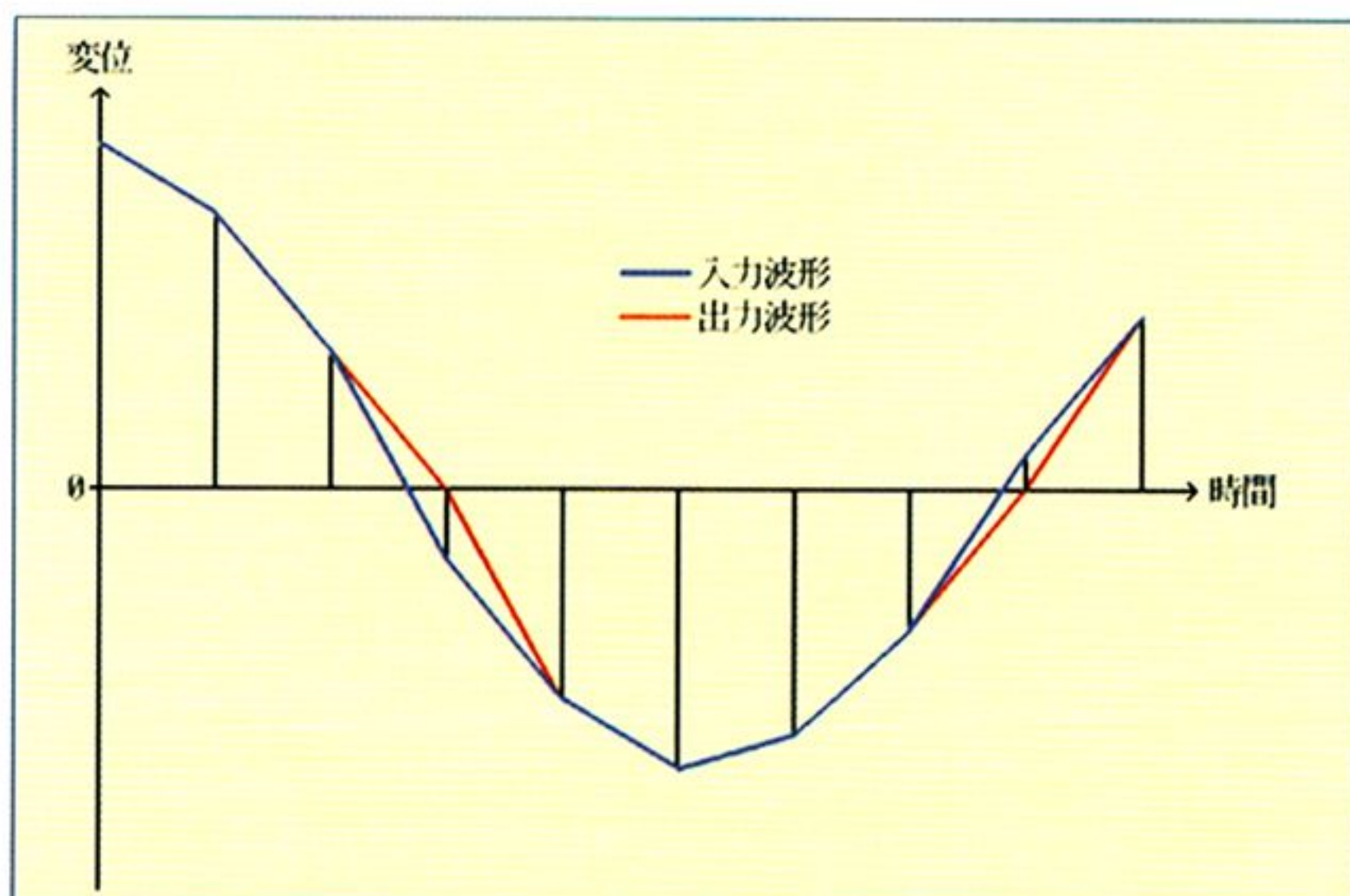



図7　符号が変わるとき変位が0に近い側を0にする

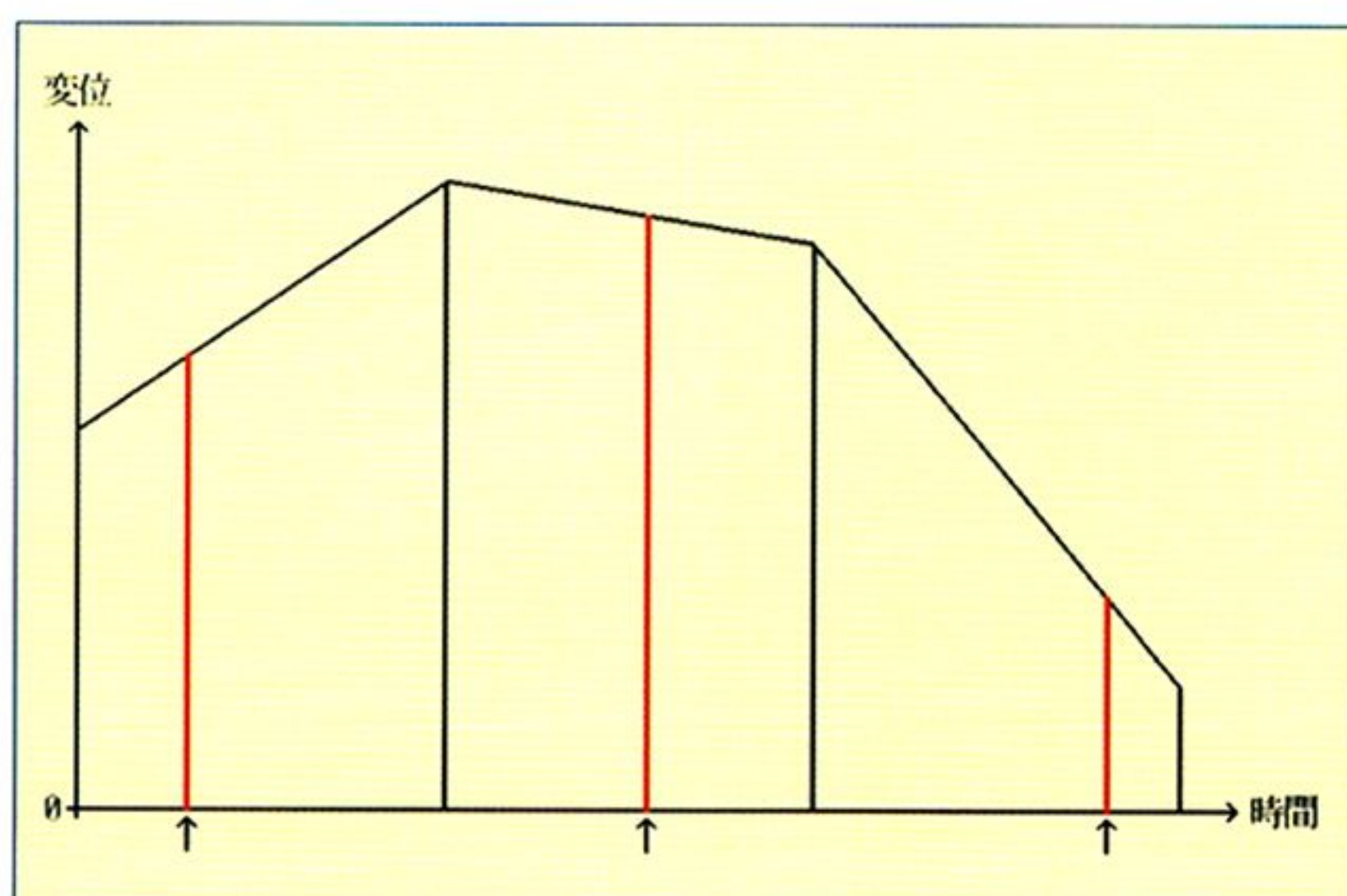



図8　サンプリング周波数の変換(直線補間)

上68060のみです。しかし，今回採用した方法の中ではもっとも音質がよいので，音質にこだわる場合はあらかじめこの方法で変換したデータのファイルを作っておくべきです。

実装

060turboで作り始めたので，.S44ファイルをオンメモリで再生できるようにすることは比較的容易でした。060turboに大容量のSIMMを装着していれば，数分間の曲をオンメモリで再生することができました。

プログラムの名前は，.S44ファイルを直接再生できるということで，S44PLAY.Xにしました。再生中はマシンのパワーをほとんど使ってしまうのでほかのことをする余裕がないことと，ディスクからデータを読み込みながら再生する機能があるため，ほかの音源ドライバのように常駐はしません。

次に，ファイルから読み込みながらの再生ができるようにしました。普通に，バッファを2つ使って，一方を再生しているあいだに他方にファイルを読み込んでおくという方法です。68030モードでも動くようになりました。ハードディスクやMOに置いてある.S44ファイルを延々と再生することができます。当然のことですが，フロッピーディスクではデータの供給が間にあいません。

初めは音楽CDからCD2PCMT.Xを使って吸い出して作った.S44ファイルを使ってテストしていたのですが，いちいち.S44ファイルを作るのが面倒になり，数十MB単位のファイルをいくつも転がしておけないので，音楽CDの音声トラックを直接読み込みながらの再生を試みてみました。CD2PCMT.Xのソースを参考にしながら作ったのですが，初めはだいぶ手こずりました。SCSIバスを何度もハングアップさせながら作っているうちに，ハングアップしても容易に復帰できるようにするための仕掛けのほうが強力になりました。X680x0がCDプレイヤーになっているさまはなかなか壮観です。

S44PLAY.Xの公開済みの最新版は激光電脳倶楽部Vol.7に収録されていますが，バグが取れていないので，月刊電脳倶楽部のほうで更新する予定です。

・フェードイン・フェードアウト

再生中はキーオンしたままなので，FM音源のエンベロープに関するレジスタを操作することで簡単にフェードイン・フェードアウトを行うことができます。フェードイン・フェードアウトの速度の指定が可能です。再生中にいつでも好きなところでフェードアウトを開始することができます。

FMPファイル

ステレオ再生時の片側のサンプリング周波数は最大でも25kHz程度までなので，低いサンプリング周波数でいかに綺麗に再生するかが重要になってきます。しかし，サンプリング周波数の変換を複雑にするとマシンのパワーが不足するのでリアルタイムでの変換ができなくなってしまいます。そこで，高品質な変換を行う場合は，サンプリング周波数の変換(およびPCMからTLへの変換)を済ませたデータをいったんファイルとして出力しておくことにしました。再生時には変換済みのデータをファイルから読み込みながらただひたすらFM音源レジスタに垂れ流すだけにすることで，特にX68030の場合はFM音源LSI(YM2151)の応答速度ぎりぎりの高いサンプリング周波数で再生できるようになりました。

リアルタイム変換で高品質なステレオPCM再生を行うには68060以上のパワーが必要です。しかし，データをあらかじめ変換しておけば68000でもディスクからデータを読み込みながらの再生が可能です。再生エンジンをゲームなどに実装するときは，あらかじめFMPフォーマットに変換済みのファイルを用意しておくことで，主題歌などを高品質のPCMで再生することが可能です。

困ったこと

・アセンブルできなくて…

S44PLAY.Xのソースリストはオールアセンブラで17000行以上あります。その中にひとつで4000行近いマクロ定義があり，マクロ内のローカルシンボルが多すぎるために従来のアセンブラではアセンブルできません。しかたがないのでHAS060.Xのほうを改造してアセンブルしています。

・SCSIの強制ソフト転送

ディスクからPCMデータを読み込みながら再生する場合，データの読み込みにDMA転送が使用されていると，MPUが割り込みを取りこぼしてしまうので正常に再生できなくなります。そのため，データの読み込みはソフト転送で行う必要があります。

X68030やMach-2のSCSI-BIOSの高レベル転送コール(_S_READ/_S_READEXT)はSRAMのソフト転送フラグに対応しているので問題ないのですが，SUPERからCompactXVIまでのX68000の内蔵SCSI-BIOSの高レベル転送コールはソフト転送に対応していないので，これらのコールを自前で展開して低レベルのソフト転送コールを使うようにする必要がありました。

・040turboには酷？

X68030と060turboの030および060モードでは25kHzで問題なくステレオ再生できます。ところがどういうわけか，040turboの040モードに限ってステレオ25kHzで正常に再生できないのです。68030よりも68040のほうが速いはずなのに……。

これはどうやら040turboの040モード時のバスアクセスが極端に遅いことが原因のようです。060turboの060モードでもバスアクセスは030モードよりも遅くなります。しかし，50MHzの68060はバスアクセスの遅さを補って余りあるパワーを持っているのに対して，25MHzの68040ではそれを補いきれないのです。こればかりは，割り込み間隔を伸ばす以外に対策がなさそうです。

おわりに

S44PLAY.Xは，X680x0用のプログラムのなかでもっともFM音源を酷使するプログラムのひとつといえると思います。なにしろ，FM音源LSI(YM2151)をこれ以上速く制御できないという速度で長時間アクセスし続けるのですから。

PCM音源とは到底呼べないような音源を使ってPCMデータを再生するという発想は，パソコンがまだ8ビットだった時代からあったものです。古くからのOh!Xの読者ならば，あの「サンダーフォース！」の“声”が記憶に残っている人も少なくないのではないでしょうか。

パソコンを特定のソフトを動かすためのプラットフォームとしてだけでなく，パソコンそのものをオモチャとして使う，しかもほかの人が思いつかない，あるいは思いついても誰もやらないような突拍子もない使い方をする，そんな楽しみ方をこれからも続けていきたいものです。

変位
0
時間

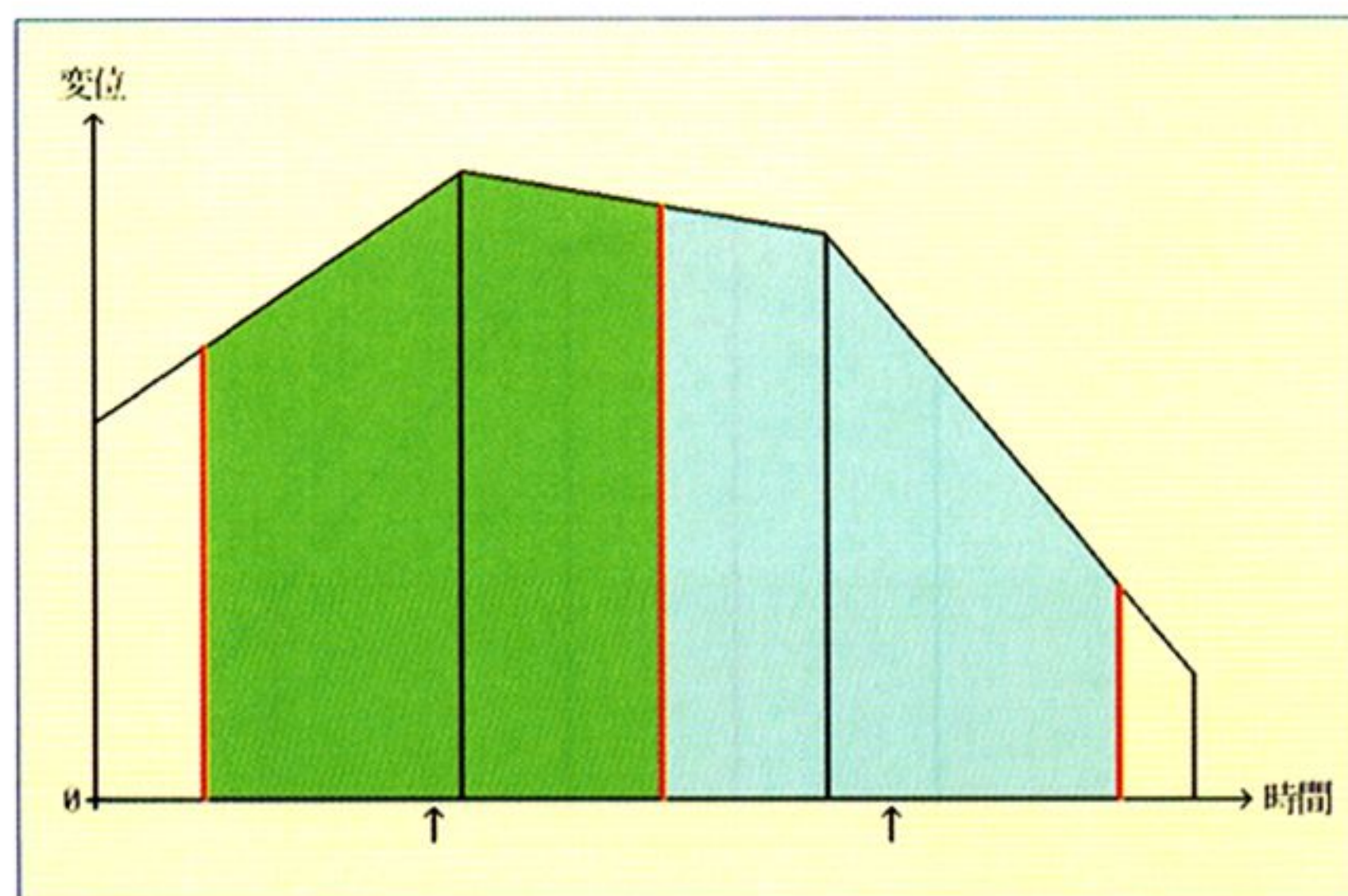



図9　サンプリング周波数の変換(面積補間)

修理して使うX68000

中村隆生　Nakamura Takao

column

本誌の読者の中には，元X68000ユーザーという方も大勢いらっしゃることと思います。そして，なかには「いつの間にか電源が入らなくなってしまった」というような方もおいでではないでしょうか。ここでは，X68000でよくある故障と，その対処方法について考察してみたいと思います。

難修理物件

たとえば画面に予期しない変な線が出る，ドットが化けるなどといった，見るからに難修理そうな物件は，素人修理では直らない可能性がきわめて高いばかりでなく，下手にいじるとかえって傷口を広げてしまい，手の施しようがなくなることもあります。

難修理物件の場合，自分で直そうとせず，シャープさん(正確には，シャープドキュメントシステム株式会社)に修理を依頼するのが確実です。幸い，まだX68000の修理は受け付けられるようですが，修理が完了するまでに時間がかかります(具体的な期間と費用については見積もりしてもらうとよいでしょう)。また，改造機は修理を拒否されることがほとんどです。

本体の軽度な故障

元祖やACEなど，初期の機体ではそろそろ機構的に寿命を迎えるものが出てくると思います。特にX68000ユーザーの多くは「改造して当然」というぐらい，クロックアップをはじめとする改造を加えている場合が多いようです。使用頻度にもよりますが，クロックアップすると，その分LSIにかかる負荷が大きくなり，寿命は縮まります。特にMPUが真っ先に寿命を迎えるようです。動作が不安定な場合，MPUを交換してみるという対処法は試してみる価値があります。

X68000の場合，MPUが死ぬと電源制御ができなくなりますので，電源が入らないなどの深刻な症状となることもあります。ただし，電源ユニットが故障しているというケースもありますので，即断はできません。本体背面の電源スイッチを入れても，POWERランプが赤く光らないという症状は，電源ユニットが故障しています。電源がちゃんと入るように見えても，電源スイッチを切った際に，LEDがいきなり消灯してしまうのにファンが回りっぱなしになるという症状は，電源ユニットではなくMPUか電源制御周りの故障であると考えたほうがよいでしょう。

ツインタワー機の場合，組み立てをミスすると，基板のハンダ面にシールドが接触し，似たような症状(電源スイッチを切るとLEDが消灯するが，ファンは回りっぱなし)になることがあります。「ぴー」という高い音が聞こえることがありますが，この場合はたいていどこかでVccとGNDが接触しています。すぐに背面の電源を切ってください。

まれに，起動時に白窓エラーが頻発するという症状を見かけますが，意外にもSCSI系統のヒューズが焼き切れているだけということがあります。ヒューズは，SCSI内蔵機種であればSCSI端子がついている基板に，SCSIボードであればSCSI端子の近くにあります。この部品はターミネータ用の電源線(TERMPOWER)の過電流を防止するために設けられているものですが，本体と周辺装置の電源が入ったまままSCSIコネクタを斜め差しすると，焼き切れてしまうことが多いようです。ひどいときには，ヒューズだけでなく，逆電流防止用に設けられている底面基板のダイオードが焼けてしまうことがあるようです。めったにないことですが，過去1例だけ見かけました。ヒューズを交換しても症状に改善が見られない場合は，チェックしてみるとよいかもしれません。

なお，交換するヒューズとしては1A即断型を使用しましょう。修理用部品としてシャープドキュメントシステムさんから取り寄せが可能です。

Compactの場合，起動しない症状が出ていた機体で，実は背面のFD番号設定スイッチの誤りが原因だった，ということがありました。

話がそれますが，ジャンク品として売られているX68000の中には，MPUソケットのうち数ピンが壊れているだけだったり，クロックがハンダづけ不良または接触不良であったりといった，比較的症状が軽いものが散見されます。こういった廉価なジャンク機を，部品取り用に購入する方もいらっしゃるようです。

電源の故障

電源ユニットが寿命を迎える場合も散見されます。あるコンデンサの寿命により故障が誘発されるというのが原因のようです。

この場合，電源ユニットを新しいものと交換すれば直るのですが，あまり安いものではありません。幸い，AT互換機もATXになってようやく電源制御が取り入れられました。これを流用できないかという提案がユーザーさんからありましたので，満開製作所で「ATX電源接続キット MK-CC011」をキット化しました。X68000全機種で使用可能です。もっとも，電源ユニットのヒューズが焼き切れている場合，ほかの部位に問題があって過電流が流れた可能性もありますので，このキットで直るという保証はできません。

ATX電源ユニットはもちろん市販品を流用します。キット側には，ATXに適合したコネクタを使っていますから電源ユニットとの接続は簡単です。

本体への給電は，もとからX68000の電源ユニットに使われていた配線を切って使用します。ですから，一部の機種では長さが足りなくなることがあります。配線はすぐわかるように，取扱説明書に明記してありますし，基板上にも配線名と配線の色をシルク印刷してあります。

X68000では最大限使っても100W程度しか必要としませんが，ATX電源は最低でも250Wと，強力なものしかありません。多い分には問題なく，電源安定のためにもいいのですが，少しもったいないという気もします。

なお，よく「電源ユニットは内蔵が可能ですか？」という問い合わせを受けるのですが，ATX電源がかなり大きいため，内蔵は無理です。ツインタワー機では，元電源ユニットがあった場所に基板が収まるような設計になっていますが，残念ながら元祖・PRO・Compactではそれほどスマートにはいきません。

FDドライブの故障

機構系の故障として多いのがFDドライブの故障です。しかし，機構系ならではのことですが，たとえばシーク不良であればグリスを塗布するだけで直るケースも見られます。たばこをよく吸う人の場合，ヘッドクリーニングも馬鹿にできない効果があります。

ドライブも車のタイヤと同様，片減りが見られます。タイヤのローテーションと同様，ドライブの入れ換えをすると良いでしょう。ドライブをよく見るとわかりますが，ドライブユニット自体は同じものが2つ使われています。ドライブ番号を設定するジャンパがあり，それだけが2つのドライブの違いとなっていますので，ユニットをばらしてジャンパを設定し直したうえで，ドライブの位置を入れ換えるとよいでしょう。

マウスの故障

マウスのボタンをクリックしても反応がないけれどもほかは正常という場合，ボタンのスイッチが故障しているだけのことが多いです。ただし，密封構造になっているのか，接点復活剤が効きません。秋葉原などでは，マウスのスイッチだけが売られていますので，この程度であれば自力で交換することが可能です。

マウスを消耗品と割り切った場合，すでにX68000用マウスは販売されていませんから，信号変換装置を介して他機種用のマウスを流用するほかありません。満開製作所ではMK-MJ2「マウズジャック2」という，PC-AT互換機用のPS/2マウスの信号をX68000用に変換するアダプタを製造・販売しています。また，同人ハードで同様の装置を作っているサークルがあるようです。

ディスプレイの故障

ディスプレイもよく経年劣化が見られるデバイスです。半固定抵抗の調整で直るうちはまだよいのですが，そのうち色ずれがしてきたり，ぼやけてきたりします。

色が，たとえばRGBのうち1系統だけ抜けるといったような場合には，ケーブルやコネクタの接触不良も考えられます。特に，ケーブルをひねると直るようであれば，コネクタ部のハンダをつけ直せば直ることがあります。

ディスプレイは分解しないに越したことはありませんが，やむをえず分解するときは，内部にシャレにならない高電圧部分がありますから，十分に注意してください。特に充電部には，基板にシルク印刷で印がしてあります。

専用ディスプレイのなかには，まだ入手が可能なものがあります。詳しくはパソコンショップ満開にお問い合わせください。また，周波数さえ対応していれば，31kHzに限ってはマルチスキャンディスプレイが流用可能なことがあります。

手抜きディスプレイの場合や，最近よく見かけるようになった「周波数の決めうち」をしているディスプレイでは，X68000の映像周波数に対応していないので，実績のあるディスプレイを選ぶようにしましょう。映ることがはっきりわかっているディスプレイであれば，中古で出物を探したほうが手堅いでしょう。

X68000のSCSI機器接続，その続きと補足

電脳倶楽部編集部
あいはらてつや　Tetsuya Aihara
aihara@mankai.co.jp

前回の補足

春号では，ざっとSCSI機器を含めたセットアップについて書いた。その後，たまたま4GバイトのHDDをセットアップする機会があった。最近ありがちな安価なATAPIのドライブにSCSI変換ボードを組み込んだタイプのドライブだったのだが，どうにも物理フォーマットがうまくできなかったので，いろいろと試してみた。

大容量デバイスを扱う場合，Human68k自体は，2Gバイトを超えるディスク空間を扱えないので，FORMAT.Xで物理フォーマット(装置初期化)を行い，そのあと論理フォーマット(パーティションをHuman68kが扱えるサイズの2Gバイト以下で区切って領域確保)したうえで，GOVERHDを使ってHDD上のIPL領域を訂正してやることでディスクとして認識させるのだが，このケースではそもそも物理フォーマットしようとするとFORMAT.Xが白帯で止まってしまうという難儀な状態にぶち当たった。

いろいろ試した末，FORMAT.Xの誤動作を回避するためSCSITAI.SYS(SCSI TARGET IOCS)を使い，SCSI IOCSの不具合にパッチを当てた状態でアクセスすることにした。こうすることで，多少大容量のアクセスに強くなることがある。

さて，今回もそれでFORMAT.Xで無事フォーマットできたのだが，物理フォーマットさえできてしまえば，あとは春号のとおりでそのまま扱えるようになった。いずれにしても，システムのバグに近い仕様を回避して使用するというリスクだけは負わなければならないので注意してほしい。特に最近の十数Gバイトのドライブではうまくいくという保証はできない。各自のリスクでお願いしたい。

それから前号の原稿を見ていて気がついたが，CD-ROMなどセクタサイズが2048バイトのドライブもあるので，config.sysのBUFFERSのところは，

BUFFERS= 20 2048

としておいたほうがいいだろう。

データ交換あれこれ

さて，今回の本題。現状ではX68000をLANに繋げるシステムを持っていない人がほとんど。この場合，データをほかのマシンとやり取りする手段としてはなにがあるか，ということについて若干の解説をしたい。以前は電机本舗から，高速なシリアル転送による擬似LAN環境を作るソフト＋ケーブルが出ていたのだが，現状ではそう簡単には手に入らないし，とりあえず，簡単なところから手をつけてみよう。

前回同様，新たに手に入れることが，極力容易であるものが前提となっている。ただし，今回は，どこで入手可能かは明記しないことにする。電脳倶楽部でもよいし，ネットにアップされているものもある。探してみてほしい。また，内部でしか使っていないものも若干あるが，それは機会があれば，満開のサイトなどでOh!X関連情報という感じで公開ということで，ひとつご容赦いただきたい。

3.5インチFD

まず，3.5インチFDという選択肢が本来あるべきなのだが，標準のX68000の3.5インチドライブでは1.44Mバイトの読み書きが実はできないのだ。これは非常に困った状態で，FDというシンプルなデータのやり取りが実は結構難しいのだ。これには3.5インチモデルが出た際，旧Oh!X編集長M氏も嘆いていたといういわくもあるくらいで，結局シャープが残した負の遺産のひとつになっている(編注：実際には，世界標準の2DD 720KBが読めないのをもっと嘆いていた)。

満開内でもすでに3.5インチのFDは，ドライブが行き渡らないので廃れている。が，1.44Mバイトの読み書きも可能な，MK-FD3計画も徐々に進行しているので，期待していただきたいところだ。ミツミ製の変哲のない3モードドライブを，機械的な仕掛けでオートイジェクトしてしまおうという，なんとも大胆な企画だ。乞うご期待といったところだ。

MOドライブ

さて，脱線してしまったが，いちばん手っ取り早い方法は，IBM Super Floppy形式でフォーマットした，MOである。X68000の世界では，あまりFDISK形式のフォーマットというのを聞かないので，国内で扱う場合にはほとんどIBM Super Floppy形式で困ることはないだろう。具体的にはSXSIのセットに添付されているIBMFORMAT.Xでも，FIM.Xでもいいのだが，これを使えばIBM Super Floppy形式でMOをフォーマットできる。このディスクであれば，WinともMacともデータ交換が行える。

市販のMOディスクは，論理フォーマット済みのディスクが多いが，論理フォーマットだけ再度かけてやれば，どのフォーマットのディスクであっても物理フォーマットからやり直す必要はないだろう。比較的信頼性も高く，可搬性もあり，ディスクサイズもいろいろ選べるので，用途によってドライブを選べばよい。満開内では230MバイトMOディスクが主流になっている。投稿データの保管やチェックもこのディスクを使っている。

他機種のデータのやり取りを行ううえでの注意点としては，ロングファイルネームが挙げられる。この場合，Vtwentyone.Xを常駐しないといけない。そのうえで，ロングファイルではエイリアス名を返すオプションを設定しておくことで，WindowsのロングファイルをIを読んだりコピーしたりすることができるようになる。また，ドライバはSusieに統一しておくのがオススメ。

またWindows，X68000間のやり取りをするディスクでは，Windows，X68000双方から書き込む操作は避けたほうがよいだろう(ひとつのディスクはどちらか一方向で)。VFATに齟齬が起きるような使い方は避ける，ということだ。これは，VFATを直接X68000上でエディットできないので壊れたときに修復が難しいためだ。

Macとのやり取りでも，設定を含めてほとんど同様であるが，Mac側はPC Exchangeを導入してないといけない。古いOSについては私もあまり明るくないが，少なくともMacOS8では問題ない。Mac上のロングファイル名は，IBM Super Floppy形式のディスクにコピーした時点で，Windows同様エイリアス名が付加される。が，ソフトによってはアクセスできないことがある。この場合は，ファイル名を勝手にX68000側で書き換えてしまうというちょっと反則な手段で読み書きしてしまうこともできる。ただし，その場合にはリソースフォークをディレクトリごと，あとで削除しておかないと，以降に同じファイル名を持ったファイルのやり取りが正常にできなくなる。

リソースフォークや，デスクトップ情報なども，PC Exchangeを使ったDOSディスク上では単なるディレクトリとファイルになる。書き込み可能なディスクにはリソースフォーク(RESOUCRE.FRKディレクトリ)が自動的に作られて，その中に各ファイルごとのリソースフォークがディスクのディレクトリ構造と同じファイル名のついたファイルとして収められている。これを丸ごと削除してしまえばなにごともなかったように，単なるDOSディスクに戻るのだが，通常のファイルのディレクトリ構造とは別にファイルシステムのカタログ構造が記録されているので，完全にまっさら，というわけではないようだ。

Macのファイル構造を保持しておきたいような

場合はやはり一方向のみの使用にディスクを分けておかなければならないが，単にファイルを移動するだけのディスクは，X68000側でリソースフォークの削除を行ってしまえば，双方向で書き込み動作を行ってもなんら問題はない。むしろ，X68000とMac間のデータ移動の問題を避けるためにリソースフォークを削ってしまうという使い方を私などはしている。

ただ，こういう使い方をしたディスクをWin-Mac間でやり取りするのは避けたほうがいいだろう。さっき述べた，カタログ構造などのデータが存在するためだ。その場合は改めて論理フォーマットしておいたほうが無難だ。

MOを使ったデータの交換は，Win-Mac間では，NTサーバーなどでネットワークを組んでデータをやり取りするほうが簡単だから，あくまで，X68000対Win，X68000対Mac間のローカルなファイル交換の際にしか，MOなどメディアを通したファイルのやり取りは有効ではない，といったところだろうか。

その他のメディアのケース

さて，その他のメディアでのやり取りとなると，やはり，大容量メディア。MOに関しては，1.3Gバイトのものも出てきているが，DOSで使う場合には制限もあるようで，あまりおすすめではない。かといってJipは接触型ということもあって，満開の中では評価は低くデータ移動用には使われていない。もっともMOが揃ってしまっていたという状況もあるのだが。

概してIOMEGA製品はWin，Macごとに専用のドライバなどが必要なものがあるのでデータ交換用には使いにくいところがある。

ただ，Jazドライブは意外に面白い使い方がある。メディア，ドライブともに信頼性が低いので，最近はあまり使っていないのだが，Jazの1Gバイトメディアを，IBMフォーマットしておくと，MacOS8.1のPC Exchangeでは，扱えないと表示されてマウントを拒否されてしまうのだが，ディスク内のユニットキャパシティを書き換えて，つまり，1Gバイト(クラスタサイズが小さくなるので)以下のディスクのフリをさせると，なんとなにごともなかったようにIBM Super Floppy形式のDOSディスクとして認識し，そのままX68000のデータを読み書きできてしまうのだった。ただ，その際に，リソースフォークのデータを書きにいってしまうので，CD-ROM用のマスターなどでファイルが多いとマウント時にはちょっと待たされてしまうことになる。しかし，こういうちょっとしたことで，なんとかなってしまうということは意外に多いのだ。ぜひ，皆さんも読めない書けないという前にSCSIについて改めて勉強してしてみるというのもいいだろう。特に，SUSIEは，ドライバの基本構造がしっかりしているので，これを改造することで読めるようになる，書けるようになるデバイスは結構多いものだ。

ちなみにFDISK形式がきちんと読めるように改造してあるSUSIEを使ってオリンパスのデジカメCAMEDIA C2000Zのスマートメディアと PCカードを使ったのだが，SCSIのPCカードリーダでFDISKとあっさり認識し，画像もJPEGファイル(ヘッダを見るとExifのようだが，中身はJFIFだったようだ)で，まったくなんのファイル加工もなしにX68000で見ることができたりしたのだが，こうしたことは，SCSIに明るければ結構なんとかなってしまうものなのだ。

DVD-RAM

また，最近はDVD-RAMドライブの価格も下がってきているが，こちらはすでにSUSIEでの対応が進んでおり，WindowsのFDISK互換の論理フォーマットをすることで，X68000では，2Gバイトメディアとして使えるようになっている。この場合は，FDISKなので，標準的なWindows環境でも読み書き可能になる。

X68000用に公開されているFDISK互換フォーマッタであるDVDFMT.Xでは論理フォーマットに機能が限定されているので，X68000のほかにWindowsマシンを持っていないと，物理フォーマットの状態チェックなどができないので注意が必要だ。光相変化書換型のメディアは書き込み回数もあまり多くなく，主にバックアップ手段にしたほうが無難だろう。

ドライブのほうでは，LUN(Logical Unit Number)によってCD-ROMとDVD-RAMを切り替えているので，X68000で使う場合にはメディアを入れていない状態でDVDSUBというパッチプログラムによってドライブのLUNを切り替えることで両方のメディアに対応できる。LUNを使えばCD-ROMとDVDで別々のドライブ名をつけて個別にアクセス可能になる。

まあちょっと高価だがCD-ROMとしても使えるし，今後いろいろなマシンのバックアップを考える場合，比較的悪くない選択肢と思われる。

ただ，X68000，Mac間では現段階でDVD-RAMの互換性はとれていない。MacではFDISK形式のフォーマットを未フォーマットと認識してしまうし，IBM SUPER FLOPPY形式ではフォーマット(マニュアルにはフォーマットできると書いてあるのだが実際には当方の環境ではうまくいかなかった)できたとしても互換性はとれなくなってしまう。

まあ，ドライブ本体はWindows版を買って，Mac版のドライバとフォーマッタを別途購入するというのもいいだろう。ただその場合，ソフトは，B's Crew liteで，MacOS標準のHFSと，HFS+でしか読み書きできていない(9月にはUDF対応のドライバ込みで正式な製品がリリースされる予定)。

もっとも，UDFはX68000で読み書きするドライバがないため，現状では使えないのだが。

ちょっとお買い物雑誌クサいが……

さて，実際にファイルはやりとりできるようになったとする。で，実際どの程度そのファイルを使えるのか，コンバートソフトなどがあまりいらない少し簡単な分野を見ておこう。

標準的なファイルに関しては，ほとんどの場合，なにもしなくてもX68000に持ってきたり持っていったりできる。

たとえば単純なテキストファイルでは，改行コード以外はほぼ問題なくコンバートできる。改行コードは，WindowsやDOSの場合，コードでいうと，CR+LF($0D,$0A)となるので，双方で問題はない。ただ，マイクロソフトのWordなどでは，テキスト形式(.txt)で書き出したり……，といった手間を要することがある。

Macの場合，通常テキストファイルの改行は，CRだけの1バイトコードになる。こうしたファイルを直接X68000用のエディタなどで読んでくると，この1バイト改行を考慮に入れていないエディタでは，正常に読めなくなることがある。ただし，標準でついてくるED.Xはそのあたりの考慮もしてあるので，一度読み込んで，再度保存してしまえば，CR+LF改行のファイルに自動的に変えてくれるので，兼Macユーザーの人は覚えておいて損はないだろう。

UNIXの場合，LF改行というテキストがあるが，これは正常に読めるだろう。

また，音声などは非圧縮の.WAVファイル(ちと大きいが)を使うことができる。これはまーきゅりーゆにっとがないとファイルの持ち腐れになると思うが，WAVをS44(ヘッダを取り除いてデータをビッグエンディアンに引っくり返せばいいので，プログラム的にも簡単だが)にコンバートしてしまえば，あとは例のS44PLAYを使って内蔵FM音源用にコンバートして鳴らしてしまうということもできる。もちろん演奏に必要なのはX68000本体のみだ。まあ，各種音声コンバートソフトは入手可能だろうから，いろいろと試してみてほしいところではある。特に，WAVファイルは，最近はMacの波形編集ソフトなどでも読めるので，

DAT(高性能ADコンバータ)→
まーきゅりーゆにっと→
WAV化→AIFF化
オーディオシーケンサで編集

といった使い方もデジタルデータの劣化がなく，非常に使える組み合わせになっている。非力なX68000といえども，音場処理のテストのようなことはできるし(もっともリアルタイムというわけにはいかないが)今後のデジタルオーディオ編集のネタとして，結構面白い分野だし，実際スタッフ内でも暇を見て研究に勤しむ人がいるほどだ。

画像といったら，もうBMPファイルだろうが，JPEGファイルだろうが，GIF，PNGに至るまで，持ってくればたいていはその形式のローダで読める。もっとも，BMPファイルなどでは，特殊な圧縮方式でセーブされている場合に読めないものもあるのだが，単純に別のマシンを持っているなら，適当なフォーマットで再セーブしてしまえばよい。今回の話はほとんど2つのマシンを持っていることが前提なので問題ないレベルだろう。

＊　＊　＊

という具合に基本的なケースを見てきた。今後はUDFか？　などと考えてしまうのだが，なかなか満開のスタッフも時間がとれなくなってきているので，各自で勉強することを望む，といったところで今回は終わりたい。

X68000の画像表示機能を考える

舩本昇竜　Shoryu Funamoto

次世代PlayStationの演算性能が凄いのはわかった。しかし，初期のPlayStationのように，映像出力に安いアナログ部品を使いせっかくの映像が台なしになる……なんてことがないことを祈る。いくらなんでも，コンポーネントビデオ(一般でいうところのDVD)端子は標準でつかなければウソだろう。

そんなとある日，勅命を受ける。
「Oh!X用にX68kのCTRCについての記事を書いてね。じゃあ」(あいはら注：そんないい加減なお願いはしていないのだが)。うむむ，CRTCと名指しされたが，CRTC単体では少し面白みにかけてしまう。ここは，勝手に拡大解釈し，大雑把なパソコンの画面表示という点に注目してみる。

パソコンにおける画面表示の基本

ディスプレイというデバイスは，ある一定のルールに従ったケーブル内を伝わる情報(絵や文字など)を視覚情報として映し出すだけの，シンプルで受動的な装置である。しかし，見た目の動作こそシンプルだが，最近のディスプレイは，1フレームあたり，数十，数百億ビット単位の膨大な情報を表示できるまでに成長している。逆の考え方をすれば，これは「ディスプレイ表示」というアクションを起こすためには，大量の情報を相手にしてやる必要があることも意味する。また，ディスプレイ表示をするには，単純に膨大な情報量を相手にするだけでなく，「タイミング」という名前の，ディスプレイ表示のためのルールに従った，規則正しい同期信号を正確に扱い続けなくてはならない。ハッキリいって，これらは大変な作業。そんな大変な処理を引き受ける「ディスプレイ相手専門IC」。それがCRTC，そしてビデオコントローラなのだ。RAMDACなんてやつもいるが，それはまたの機会に。

ディスプレイケーブルを流れる信号たち

ディスプレイ表示に必要な情報(ディスプレイケーブルを通り，パソコンからディスプレイに向かって流れる信号)は，大きく2種類に分けることができる。ひとつめが，映像信号。画面に映し出す，色に関する情報(Red,Green,Blue：光の3原色で構成される)がアナログで送られる。もうひとつとして，同期信号。ディスプレイ表示に必要なタイミング情報がデジタルで送られる。ディスプレイはこれら2つの情報を受け取り，これらがディスプレイの表示ルールに合致していれば，画面表示を行う。ルールに合致しない場合，ディスプレイの種類によってその態度はマチマチとなる。共通しているのは，「期待どおりの画面表示」が得られない，といったことか。

注）一般的なルール違反
ルール違反の代表格といえば「範囲外周波数」だろう。具体的には，表示可能水平走査周波数30kHz～60kHzのディスプレイにNTSC解像度(水平走査周波数15kHz)信号を送るような行為である。この場合のルールは「水平走査周波数というタイミング信号は30kHz～60kHz であること」であり，X68000の低解像度信号を送るということは「15kHz という 30kHz を下回るタイミング信号を送る」というルール違反を犯している。よって，この場合画面表示が正常に行われない。

○表示開始までに

ディスプレイは，まず同期信号を受け取り，ソレがルールを守っているか，――表示許容範囲内の情報であるかどうか――を見極める。守られていることを確認できれば，画面表示のために，今度はディスプレイが，送られている同期信号に歩調をあわせようとする。

具体的には，同期信号が「画面左上の点の表示」を意味する状態になるまで，ディスプレイは，画面左上の点に色をつけられる状態(ブラウン管ディスプレイの場合，画面左上にビームを発射できる状態)で待つ。そして，同期信号が「画面左上の点の表示」を表す状態を迎えたあと，ディスプレイは同期信号のタイミングにあわせ，順次映像信号を読み取り，画面上の点の色に反映(ブラウン管の場合ビームを発射)するという動作を繰り返す。これがディスプレイにとっての画面表示プロセスなのだ。

微妙かつ明確な役割分担

さて，X68000の場合，ディスプレイ信号は大雑把に，同期信号はCRTCが，映像信号はビデオコントローラがその実権を握っていると考えてよい。

誤解している人もあるかと思うので，一応念を押しておく。CRTCとは「CRTコントローラ」の通称であり，確かに画面表示に関係するありとあ

水平タイミング制御	R00	$E80000	水平トータル
	R01	$E80002	水平同期終了位置
	R02	$E80004	水平表示開始位置
	R03	$E80006	水平表示終了位置
垂直タイミング制御	R04	$E80008	垂直トータル
	R05	$E8000A	垂直同期終了位置
	R06	$E8000C	垂直表示開始位置
	R07	$E8000E	垂直表示終了位置
水平位置微調整	R08	$E80010	外部同期水平アジャスト
ラスタ割り込み用	R09	$E80012	ラスタ番号
テキスト画面スクロール	R10	$E80014	X位置
	R11	$E80016	Y位置
グラフィック画面スクロール	R12	$E80018	X0
	R13	$E8001A	Y0
	R14	$E8001C	X1
	R15	$E8001E	Y1
	R16	$E80020	X2
	R17	$E80022	Y2
	R18	$E80024	X3
	R19	$E80026	Y3
メモリモード/表示モード制御	R20	$E80028	SIZ COL HF VD HD
同時アクセス/ラスターコピー/高速クリアプレーン選択	R21	$E8002A	MEN SA AP CP
ラスタコピー動作用	R22	$E8002C	ソースラスタ デスティネーションラスタ
テキスト画面アクセスマスクパタン	R23	$E8002E	マスクパターン
画像取り込み/高速クリア/ラスタコピー制御	CRTC動作ポート	$E80480	RC V FC VI

CRTインタフェースの基本タイミングとCRTCの標準設定値

タイミング		高解像度	標準解像度
同期周波数	水平	31.5kHz	15.98kHz
	垂直	55.46Hz	61.46Hz
データ表示期間（1）	水平	22.09μs	55.69μs
	垂直	16.25ms	15.019ms
同期期間（2）	水平	31.75μs	62.58μs
	垂直	18.03ms	16.270ms
同期パルス幅（3）	水平	3.45μs	3.308μs
	垂直	0.191ms	0.187ms
バックポーチ（4）	水平	4.14μs	4.94μs
	垂直	1.111ms	0.876ms
フロントポーチ（5）	水平	2.07μs	1.65μs
	垂直	0.476ms	0.187ms

レジスタ		画面モード（高解像度）				画面モード（標準解像度）		
番号	アドレス	768×512	512×512	512×256	256×256	512×512	512×256	256×256
R00	$E80000	$89 (137)	$5B (91)	$5B (91)	$2D (45)	$4B (75)	$4B (75)	$25 (37)
R01	$E80002	$0E (14)	$09 (9)	$09 (9)	$04 (4)	$03 (3)	$03 (3)	$01 (1)
R02	$E80004	$1C (28)	$11 (17)	$11 (17)	$06 (6)	$05 (5)	$05 (5)	$00 (0)
R03	$E80006	$7C (124)	$51 (81)	$51 (81)	$26 (38)	$45 (69)	$45 (69)	$20 (32)
R04	$E80008	$237 (567)	$237 (567)	$237 (567)	$237 (567)	$103 (259)	$103 (259)	$103 (259)
R05	$E8000A	$05 (5)	$05 (5)	$05 (5)	$05 (5)	$02 (2)	$02 (2)	$02 (2)
R06	$E8000C	$28 (40)	$28 (40)	$28 (40)	$28 (40)	$10 (16)	$10 (16)	$10 (16)
R07	$E8000E	$228 (552)	$228 (552)	$228 (552)	$228 (552)	$100 (256)	$100 (256)	$100 (256)
R08	$E80010	$1B (27)	$1B (27)	$1B (27)	$1B (27)	$2C (44)	$2C (44)	$2C (44)

らゆる仕事をこなす。さらに，設定すべきレジスタの数も多ければ，守備範囲も広い。しかし「画面表示に関するすべての実権をCTRCが握っているわけではない」ことに注意しなければならない。特にX68000に関しては，スプライトやBGに関する処理を引き受ける「スプライトコントローラ」という存在もあり，また，ビデオコントローラやその先のD/Aコンバータとの連携があり，初めて「X68000としての画面表示」が行われるのだ。

そこで，確認の意味を込めて，CRTC，ビデオコントローラのそれぞれについての基本的な役割をみることで，それぞれの役割の違いについて考えてみる。スプライトコントローラは，その存在自体が特殊で，かつ名が体を表しているので(これはこれで非常にユニークな存在ではあるものの)，ここではとりあえず説明は除外する。

○CRTCのお仕事(その1)：画面モード/表示タイミングの設定

前述したように，ディスプレイ表示のためには，無秩序に画像データを垂れ流すだけではいけない。ディスプレイが持つ「うまく表示できるタイミング」にあわせてデータを送ってやらなくてはならない。

CRTCは，基本的な各同期信号の生成や表示期間のタイミング調整用のレジスタを持ち，さまざまな値を設定することにより，複数の画面モード，異なった表示周波数や表示方法(ノンインターレース/インターレースなど)を実現できる。忘れてはいけないのは，タイミング信号を管理し出力し続けるのはCRTCの仕事なのだが，あくまで，その信号に関するタイミングを設定するのはユーザー(システム)であるということ。といっても，よほどのことがない限り，実際にCRTCのレジスタを気にするようなことはない……と書きたいトコロなのだが，特に旧Oh!X誌による「メガディスプレイ計画」記事のおかげで，X68000ユーザーほどCRTCのレジスタを直接操作するユーザーはないだろう。

注）GP11.Xについて

電脳倶楽部第60号以降に使用している「画面比1：1の6万色モード」プログラムGP11.Xは，CRTCの特性を逆手にとったプログラムといえる。

具体的には，PIC.Rなどを使い，いったんCRTCやビデオコントローラ設定を6万色モードのソレに設定し，画像をロードする。その直後に，CRTCの水平方向表示に関する設定(R00～02)の設定を768×512モードのソレに，コッソリ設定し直す。大切なのは「破綻しない程度のバランスで矛盾(？)設定」をすること。ほかにも細かな設定(水平方向の表示ドット数を512＋未定義ビット(CRTC-R20のHD=%10)の設定)もいくつかしているが，このようにすることで，ビデオコントローラは「現在，6万色モード」，CRTCは「768×512ドットモード(ただし，水平表示ドット数は512ドット)」というそれぞれの仕事をこなし，結果的に，ディスプレイには画面の表示密度は768×512ドットのソレで，6万色の512×512ドット表示が得られるのだ。

ということで，以下にCRTC-R3の値の算出方法を示す。

$$R03=\frac{((\text{水平同期期間})+(\text{水平フロントポーチ}))\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8}-5$$

より，それぞれの値を代入すると

ここが768でないことに注目！(→ 512)

$$R03=\frac{((31.75)+(2.07))\times(512)}{(22.09)\times 8}-5$$

R03=97.9846084201－5→$5C

○CRTCのお仕事(その2)：グラフィック/テキストの表示位置(スクロール)指定

グラフィック/テキスト画面の表示開始位置の指定(ハードウェアスクロール)および，それにともなうグラフィックの球面スクロール(天地左右折り返し処理)，テキストの円筒スクロール(天地折り返し処理)を管理。2Dシューティングゲームにおけるスムースなスクロールや「はみ出した部分を折り返して表示する」という処理はCRTCが行う。

これらは，X68000ユーザーが当たり前のように使う，というより暗黙の了解的に行われる処理であるが，一般的な(というより一般のアマチュアが，なんとかすれば入手可能な)CTRCにはこのような「高級」な機能はない。ありがたく使うよーに。

CRTC R00～R07の設定値の算出法

$$[R00]=\frac{(\text{水平同期期間})\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8}-1$$

$$[R01]=\frac{(\text{水平同期パルス幅})\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8}-1$$

$$[R02]=\frac{((\text{水平同期パルス幅})+(\text{水平バックポーチ}))\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8}-5$$

$$[R03]=\frac{((\text{水平同期期間})+(\text{水平フロントポーチ}))\times(\text{水平表示ドット数})}{(\text{データ表示期間})\times 8}-5$$

$$[R04]=\frac{(\text{垂直同期期間})}{(\text{水平同期期間})}-1$$

$$[R05]=\frac{(\text{垂直同期パルス幅})}{(\text{水平同期期間})}-1$$

$$[R06]=\frac{(\text{垂直同期パルス幅})+(\text{垂直バックポーチ})}{(\text{水平同期期間})}-1$$

$$[R07]=\frac{(\text{垂直同期期間})-(\text{垂直バックポーチ})}{(\text{水平同期期間})}-1$$

スクロールレジスタのワナ？

X68000はグラフィック16色4画面モードを持っているため，グラフィック画面用スクロールレジスタは4組存在する。16色，768×512ドット1画面モード(実画面1024×1024ドット)時においては，有効なスクロールレジスタは1組(R12,R13)だけとなり，残りのレジスタは無視される。

しかし，同じ1画面でも6万色モード時においては，残りのレジスタを無視するコトができず，4組のスクロールレジスタすべてに同じ値を入れなければならない。通常のスクロールはもちろんのこと，ラスタスクロールの場合も同様である。もちろん，このルールを逆手に取った特殊効果も存在する。

また，CRTCのスクロールレジスタはすべてWRITEオンリー。つまり，現在値をCRTCのレジスタから読み出すことができない。つまり，画面がスクロールするゲームなどを作る場合は，スクロールレジスタの値をセットする前後のどちらでもよいので，ユーザー自身が管理可能なワークエリアに表示開始位置を毎回保存しておかないと，あとで困ることになる。

○ビデオコントローラのお仕事(その1)：実画面サイズ/色数の制御

グラフィックの16/256/6万色モードの指定や，実画面サイズ(512×512/1024×1024)を指定するのは，ビデオコントローラの仕事である。意外に感じる人も多いと思うが，「実画面の管理はビデオコントローラ」「実画面内の表示域の管理はCRTC」と覚えておくといいだろう。

あと，忘れてはいけないのが，パレットの制御という仕事。これも実はビデオコントローラの仕事であり，この兼ねあいでグラフィックの色モードをビデオコントローラが管理している，と覚えるといいだろう。

パレットについて

X68000のメモリマップを見れば一目瞭然なのだが，X68000で使用するパレットはすべてビデオコントローラ用に割り振られているアドレス内に設定されている。

ここで，重要なのが「スプライトやBGのパレットもビデオコントローラが管理している」ということ。まさに，X68000の色に関する中枢といっていいだろう。

あと，よほどのことがない限り気にする必要は

0 水平偏向周波数 15.98kHz
1 〃 31.5 kHz

0 0 垂直256ドット
0 1 垂直512ドット
1 0 未定義
1 1 未定義

16色モード 0 0
256色モード 0 1
未定義 1 0
65536色モード 1 1

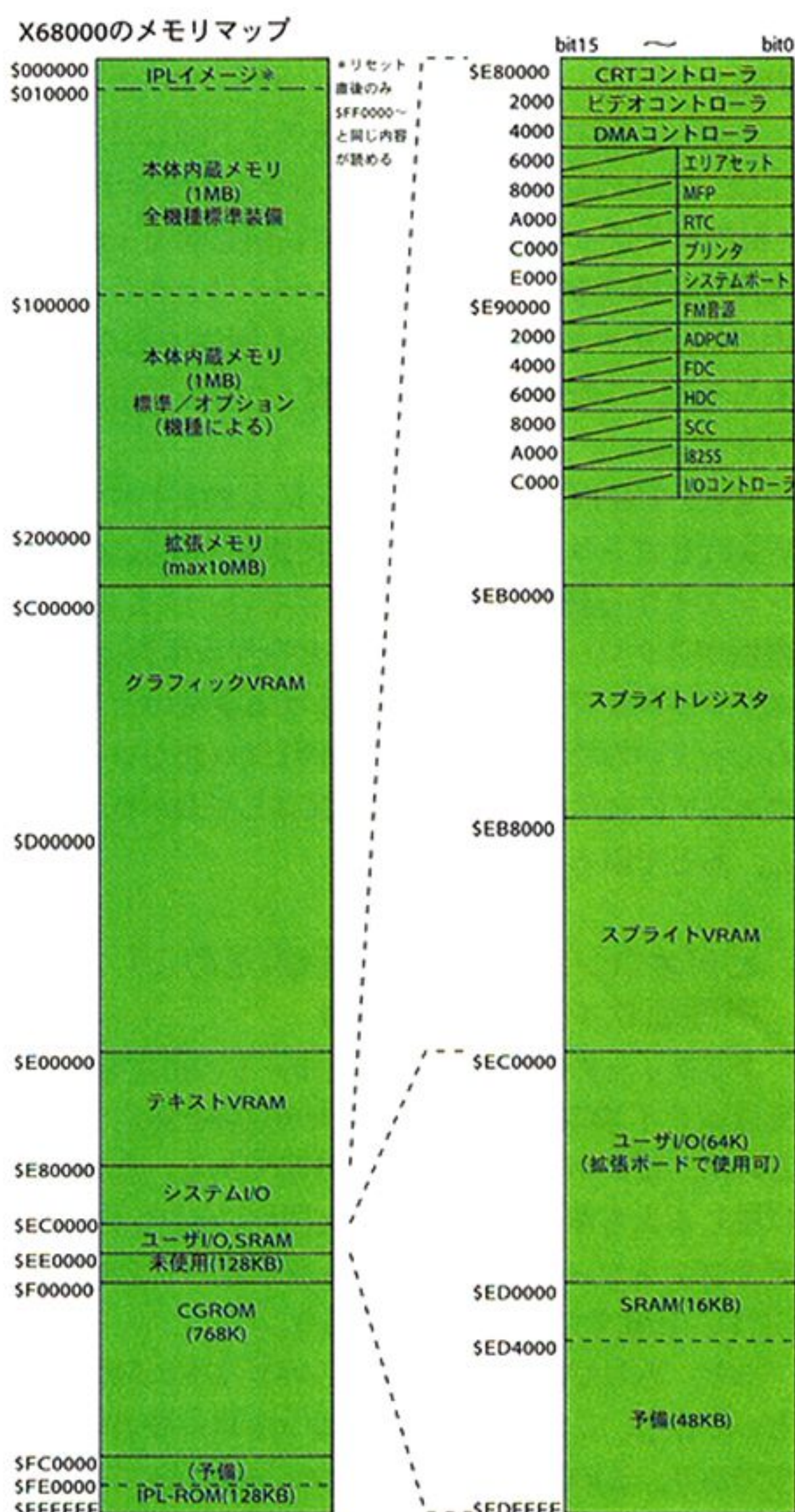
0 0 水平256ドット
0 1 水平512ドット
1 0 未定義
1 1 水平768ドット

実画面512×512ドットモード 0
実画面1024×1024ドットモード 1

bit15 10 9 8 bit0

SIZ COL HF VD HD

ビデオコントローラのR0(アドレス:E82400H)の下位3ビットを同一にすること

ないのだが，6万色モード時にもグラフィックパレットは有効であること。触らぬ神に祟りなし，といったトコロか(編注：SION IVでは，一定パターンの色を絶対に使用しないという条件下で6万色モード時にパレットを使った特殊効果を多用していた)。

X68000のメモリマップ

アドレス	内容
$000000	IPLイメージ*
$010000	本体内蔵メモリ(1MB)全機種標準装備
$100000	本体内蔵メモリ(1MB)標準／オプション(機種による)
$200000	拡張メモリ(max10MB)
$C00000	グラフィックVRAM
$D00000	
$E00000	テキストVRAM
$E80000	システムI/O
$EC0000	ユーザI/O,SRAM
$EE0000	未使用(128KB)
$F00000	CGROM(768K)
$FC0000	(予備)
$FE0000	IPL-ROM(128KB)
$FFFFFF	

*リセット直後のみ$FF0000～と同じ内容が読める

アドレス	bit15 ～ bit0
$E80000	CRTコントローラ
2000	ビデオコントローラ
4000	DMAコントローラ
6000	エリアセット
8000	MFP
A000	RTC
C000	プリンタ
E000	システムポート
$E90000	FM音源
2000	ADPCM
4000	FDC
6000	HDC
8000	SCC
A000	i8255
C000	I/Oコントローラ
$EB0000	スプライトレジスタ
$EB8000	スプライトVRAM
$EC0000	ユーザI/O(64K)(拡張ボードで使用可)
$ED0000	SRAM(16KB)
$ED4000	予備(48KB)
$EDFFFE	

○ビデオコントローラのお仕事(その2)：プライオリティ制御/半透明処理

グラフィック/テキスト/スプライト(BG)の重ね合わせ優先順位や表示のON/OFFを設定。また，グラフィック·テキスト間の半透明処理等，最終的な画面出力に必要な設定の多くを行う。

注）グラフィック画面の特殊プライオリティ？

グラフィック1画面モード時に，本来設定を行う必要のないグラフィックページ1～3のプライオリティをデタラメ設定すると，役に立ちそうもない効果が得られる。

6万色モード時においては，パレットブロックの入れ替えが行われたような(色がバグッたような？)効果が得られる。表示されている絵にもよるが，1ワードの書き換えで(指定どおりの変化はできないモノの)画面全部の色を変化/復帰ができる。せいぜいフラッシュ的効果が関の山か？　また，実画面1024×1024ドットモード時においては，4ページ分のプライオリティを全部0に設定してやると，512×512ドットの表示が2×2の状態で表示される。街頭ディスプレイによくある沢山のテレビを縦横に積み重ねたような雰囲気があり，なかなか面白い。これまた役に立つかどうかは不明だが。

グラフィックパレットの機構（16／256色モード時）

bit7 ～ bit0: VRAMのデータ　256色モード

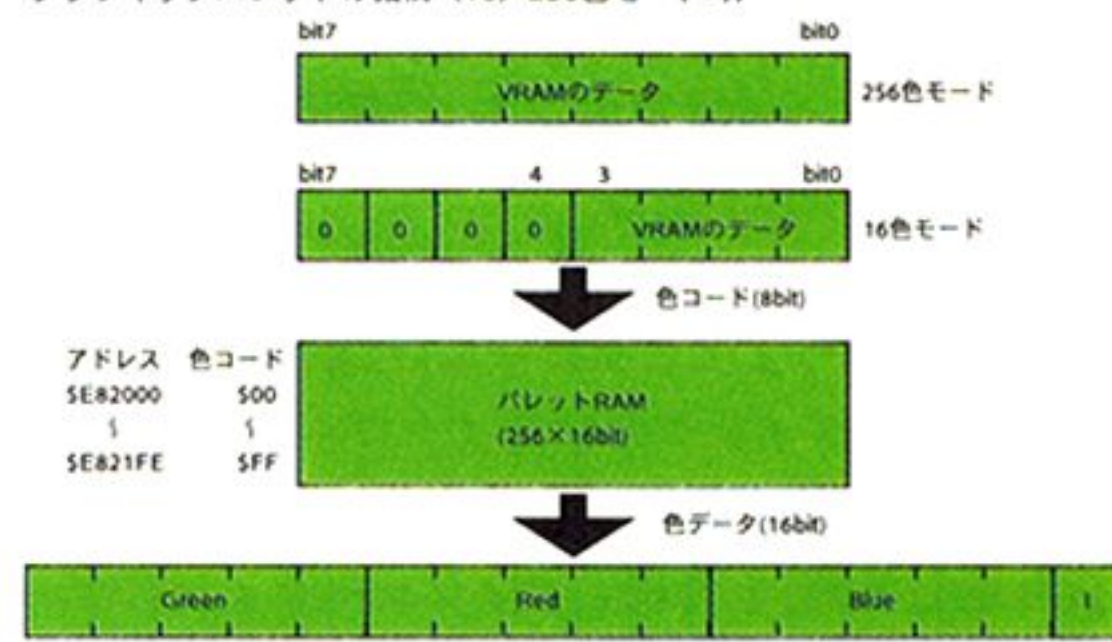
bit7 ～ 4: 0 0 0 0 / 3 ～ bit0: VRAMのデータ　16色モード

↓ 色コード(8bit)

アドレス	色コード
$E82000	$00
～	～
$E821FE	$FF

パレットRAM(256×16bit)

↓ 色データ(16bit)

Green / Red / Blue / I

○他機種でも画面表示の基本は一緒

ディスプレイに映し出される「目に見える情報」はビデオコントローラが管理しているのだが，そもそもビデオコントローラはなにを目に見える情報として管理するのだろうか。その答えは「VRAMの内容」。

VRAMとは，Video-RAMの略称で，文字どおり「画面表示用メモリ」という意味で「ブイラム」と発音する。VRAMは通常のメモリ空間と同じようにプロセッサからアクセスされ，その内容を自由に書き換えることができる。ビデオコントローラは，このVRAMに格納されている「データ(色の番号)」を読み取り，「ディスプレイが理解できる色データ」に変換し，ディスプレイに送ってやるのが仕事。たとえば，VRAMが赤色のデータで埋め尽くされた場合。ビデオコントローラは「ディスプレイが赤と理解する色データ」をディスプレイに送り続ける。ディスプレイは「赤色と解釈できるデータを表示しろ」との指示に従い，画

テキスト、スプライト＋BG画面用パレット

アドレス	色コード	色データ G	R	B	I
$E82200	$00				
$E82202	$01				
$E82204	$02				
:	:				
$E8221E	$0F				
$E82220	$10				
$E82222	$11				
:	:				
$E823FC	$FE				
$E823FE	$FF				

$00～$0F：テキスト画面で使用
$00～$FF：スプライト＋BG画面で使用可

表示サイズと人間心理

column

もう6年も前の話なのだが，電脳倶楽部誌上で初めて「CGのドット比1:1の6万色モード」表示を行ったときのこと。「表示画面が正方形ではなく，少し縦長になるのですが」との報告が予想以上に多く，少し驚いた覚えがある。

これはなにを意味しているかというと，X68000の通常画面(COMMAND.Xなどのコマンドシェル上でED.Xなどのテキストエディタを動かす場合を想定:768×512ドットモード)のドット比が縦長に設定されている，ということなのだ。この現象のカラクリそのものは実にシンプルなのだが，実はかなり根の深い問題をかかえている。いまさらではあるが，この問題に触れてみよう。

そもそも，パソコンの映像出力信号には，ドット比に関する情報は含まれていない。少々乱暴ないい方をすれば，X68000の768×512モードの場合，X68000は1フレーム(1画面)あたり393216ドット分のRGB情報をディスプレイに単純に垂れ流しているにすぎない。また，その情報がどのようにディスプレイに映し出されるかなど，X68000本体に知る術もなく，実際の画面表示状態は人間が確認するしかない。

そこで定規を用意してもらいたい。まず，コマンドシェル上でテキストエディタを起動し，実際にディスプレイに映し出された表示画面のサイズを，定規で測ってほしい。シンプルに物事を考えれば，表示画面全体の縦横サイズ比は，768ドット対512ドット(3:2)になるべきであろう。

15インチモニタを使用している場合，横のサイズは約26センチメートル。ということは，縦のサイズは約15センチメートルとなればいいわけである。ディスプレイの垂直振幅(ツマミなど)で画面表示サイズを調整してもらいたい。そうすると，このとき表示される1ドットに注目すると，そのドット表示サイズは縦横ともに同じなる。つまり，表示ドット比1:1となる。

しかし，ここからが問題かつ根の深いトコロ。それは，15インチモニタの場合，垂直振幅を操作すれば，縦のサイズを約19cm程度まで引き延ばすことができる。つまり，「本来あるべきサイズ」ではなく「とにかく画面いっぱいに表示」することもできるのだ。それがたとえX68000の768×512ドットモードであっても，ディスプレイ側の調整で，約26×15cm(3:2)であるべき表示サイズを約26×19cm(4:3)にまで，変更することができるのだ。

逆の考え方をしてみよう。本来あるべきサイズにディスプレイを設定した場合，15インチディスプレイユーザーは，常にディスプレイ上下に2センチメートルの余白を強いられることになる。数字を読む限りではたいしたことはなさそうだが。実際問題として，19センチメートル中の上下2センチメートルという空間は相当大きく，人によっては単に「なにも表示されない空間がもったいない」，また人によっては余白によって「文字が(実際にはそうではないが)上下に潰されるという圧迫感」を感じ，精神衛生上よくない。

さらに問題は深刻化し，特にテキストエディタの起動率が高いX68000ユーザーの768×512ドットモードは，「表示サイズ比3:2は縦に潰れている」と「表示サイズ比4:3は縦長すぎる」のイメージの間をうろつき，結局なんだか複雑な表示サイズ比(16:11など)になるケースが多いようだ。

なにもこういった問題はX68000に限ったわけではない。最近のPC/AT用ビデオカードにおける，1280×1028ドット表示などはいい例だろう。定規で実際に表示サイズを測り，縦横の実測サイズ比が10:9という人の率はどのくらいいるのやら。

面に赤色を出力し続ける。

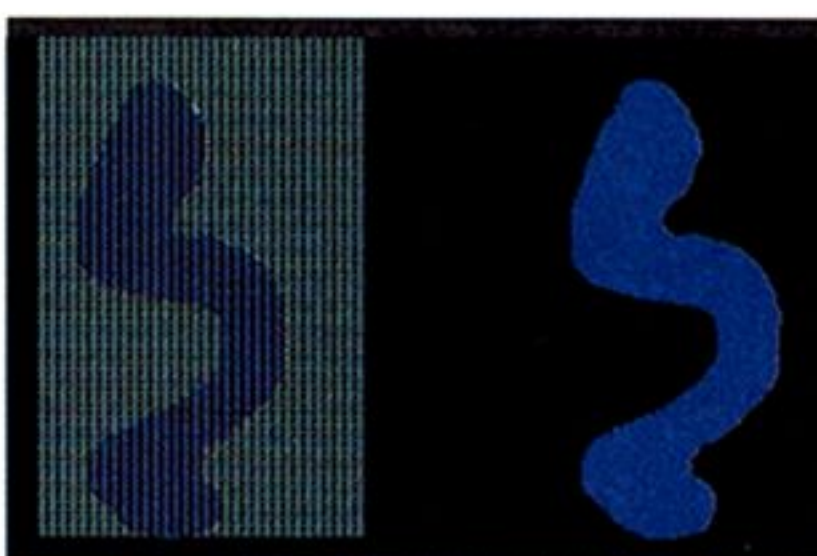
○縁の下のチカラモチ

以上のように,「色」や「目に直接見える」情報に関わりを持つのが,文字どおりビデオ(Video)コントローラ。ディスプレイ(CRT)のタイミング情報に関わりを持つのが,これまた文字どおりCRTコントローラ。と覚えておくとよいだろう。

世の中のグラフィックカードが,「VRAM容量がウンタラ」という面ばかり強調されていることを考えると。CRTCやビデオコントローラは目に見えないトコロで活躍する黒子といったトコロだろうか。

注)緑子(本当の雑談)
ハリウッドでは,映画製作にCGが用いられるようになってからブルーバックならぬ,「グリーンバック」という撮影方法が流行っているという。文字どおり,緑一色のスタジオの中で演技し,CGワークで必要な部分(緑色の部分)だけを合成するのだが,ブルーバックに比べ,明るい背景を合成するのに適しているという。さて,そのグリーンバック撮影時に活躍するのが黒子ならぬ「緑子」。全身緑の人間が背景の緑に溶け込むような状態で人形を動かし,新作のスターウォーズのC-3POも緑子が動かしているという。

短形領域に分割する方法

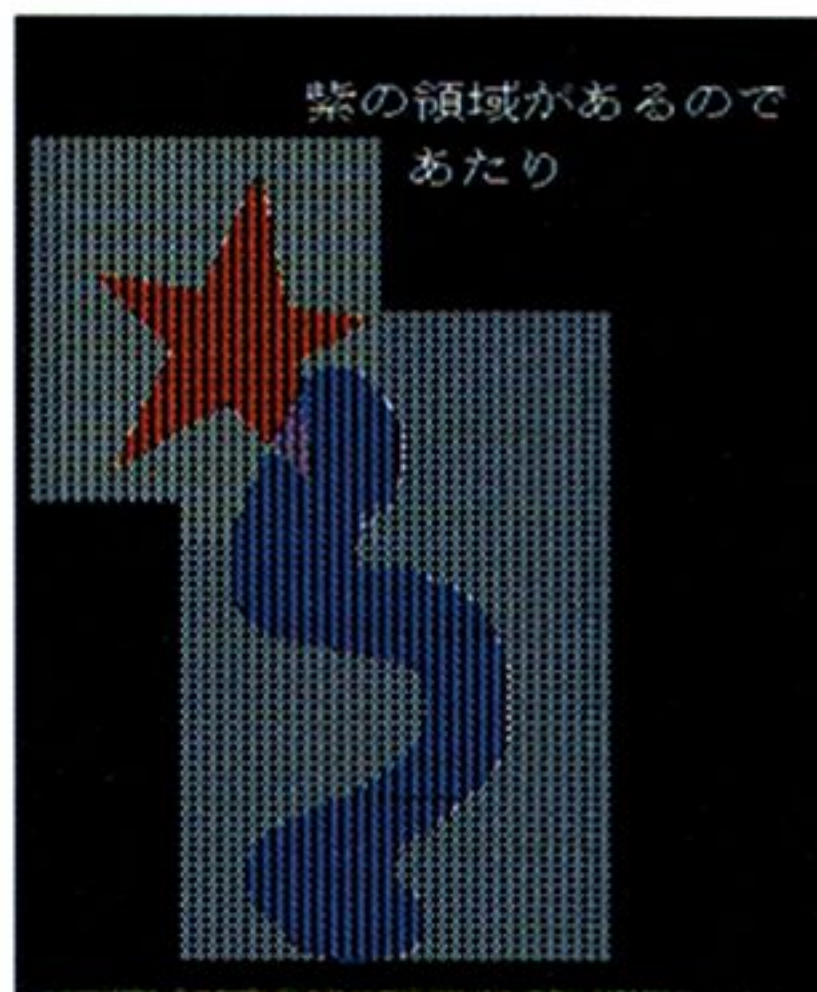
パターン部分と透明部分をコード化する

紫色の部分でぶつかっている

外枠は重っても衝突はしていない

○表示用コントローラを応用した当たり判定

『首領蜂』あたりから本格化し『怒首領蜂』で開花した「当たり判定は小さくてナンボ」の風潮。現在,正確な当たり判定は,あまり必要とされなくなってきた感がある。それでも,なかなか興味深い話であると思うので,その昔,アーケードゲームに搭載されていた,当たり判定機能について触れてみたい。

・海のなかでは?

たとえば「わかめ」キャラと「ひとで」キャラ同士の当たり判定を考えてみる

いってみれば,「☆」と「ぐにゃぐにゃ」の当たり判定。いくつかの矩形領域で近似できなくはないだろうが,それっぽい判定を行うには,相当な数の矩形領域を用意する必要があり,やってやれないことはないが,ソフトで記述する手間は非常に大きい。

・下準備

まず,わかめとひとでのそれぞれのパターンデータについて,2値のビットマップを用意する。つまり「パターン(色)のある部分=1/透明部分=0」のデータをキャラクターパターンとは別に用意し,これを「ヒットチェックビットマップ」と呼ぶ。説明の簡略化のために,ここでは,わかめのヒットチェックビットマップをスプライトコントローラの「赤ヒットチェックグループ」,同様にひとでのヒットチェックビットマップを「青ヒットチェックグループ」と呼ぶことにする。

・当たり判定の実行

勘のいい人はもう気づいたと思うが,あとは,キャラクターを実際に動かし,それぞれのヒットチェックグループに登録したわかめとひとでのビットマップの重なりである「紫」の部分が現れば,キャラクターの衝突と判定される。といったカラクリである(実際には,ヒットチェックグループは8個まで持つことができるので,ルックアップテーブルを用いたマルチビットチェックのようだ)。ソフトで同じような判定もできなくはないが,ハードで高速検出するトコロに,昔のアーケード的ロマンを感じる。

ヨタ話 column

極端な話,ディスプレイ表示は「ルールに従った情報を正確に出力」できる相手となら,誰とでも可能だ。パソコンに限らず,電子レンジであろうと冷蔵庫であろうと,誰とでもつながることができ,ディスプレイの持つその「情報を目に見えるカタチで表示する」という機能を発揮することができる。問題は,そのルールそのものが複数存在し,微妙かつ複雑で,さらに高コストであること。

ルールが違うから,ディスプレイにビデオデッキはそのままつながらない。テレビにパソコンの画面を映し出すことも(普通は)できない。微妙かつ複雑なので,たいていの場合,X68000の画像信号は,最近の液晶ディスプレイに映らない。そしてなにより,高コストなので基本的にディスプレイはパソコン以外に繋げられることがない。特に,有限種類の情報を扱う場合,投資に対する見返りのバランスが非常に悪い。

冷蔵庫にディスプレイをつなげた例に考えてみる。話題の「インターネット冷蔵庫」としてではなく,また,薄いLEDパネルなどをドアに貼り付けるわけでもない。冷蔵庫にディスプレイ端子が設置され,ケーブルのその先にディスプレイがつながっている構図。

まず,純粋な冷蔵庫として,ディスプレイに表示させたい情報とはなんだろうか。各冷凍室温度の温度(時間軸を持った温度グラフでもいいだろう)や,製氷室内の氷のできあがり(お肉の解凍具合)予測時間といった数値的な情報。ドアの開けっ放しや詰め込みすぎなどの警告メッセージ的情報などが考えられる。そして,現在の技術をもってすれば,これらはすでに実現できてしまう。冷蔵庫にぶら下がった(つながった)ディスプレイでは「ビール優先状態で運転中です」「あと7分で解凍完了予定です」等々の情報が表示されるイメージ。

さて,ここからが問題。ごく普通の一般で中流家庭の皆さまにとって,このような様子はどのように目に映るのだろうか? おそらく「便利かもしれない」であると同時に「なくても全然困らない。それがどうしたの」程度というのが,正直なトコロだろう。

ドアの開けっ放し(1ビット)や詰め込みすぎ(nビット)に関しても,ケーブルを介したディスプレイの先で警告されるより,冷蔵庫本体からビービー音が鳴ったり,「ドアが開けっ放しです」とかPCMなどで喋らすほうがテットリ早い。

ようするに,「ビール優先状態で運転中です」と表示することのできる「値段の高い」冷蔵庫を一般の人がわざわざ買うかどうか。冷蔵庫にディスプレイがつながる。という構図はユニークだとは思うのだが。

1999年のスプライト

――XSPによる物理法則の事前処理――

山口光樹　Yamaguchi Mitsuki

X68000の特徴は？　と問われれば，M68000 MPU，マンハッタンシェイプ，512×512ドット65536色同時発色，オートイジェクト5インチFDD，FM音源，AD PCM音源，テレビコントロール，ポップアップハンドル……とさまざまな装備が挙げられますが，そのなかでも多くの人たちに圧倒的に支持されたのが「パソコンレベルでアーケードゲームに比肩する性能のスプライト」が装備されたことでしょう。スプライトの標準装備により多くのゲームプログラミング少年が育ち，そして大人になった少年達が現在のゲーム業界を支えていることは周知の事実です。

それから幾年月が流れ，多くのアーケードゲームは2Dから3Dへと移行し，残った2Dゲームもハードウェア的にはX68000のようなハードウェアスプライトからソフトウェアスプライト（透明色指定転送）へと変化しつつあります。確かにハードウェアスプライトを扱うという技術はすでに過去のものとなりましたが，その基本技術「垂直同期単位で指定したパターンを指定した座標に表示する」というゲームプログラミングの本質はなんら変わることはありません。そこで本稿ではゲームプログラミングの基本であるスプライトの扱い方を解説することにします。

図1　パレットによる色の変更

スプライトの仕組み

スプライトといえども，メモリマップドI/O方式を採用しているM68000 MPUでは単に指定のアドレスに任意の値を書き込むことによって制御することが可能です。スプライトを制御するICであるCYNTHIAには$eb0000番地以降のアドレスが割り当てられ，これらはスプライトレジスタと呼ばれています（スプライトレジスタの詳細に関しては文献1に掲載されています）。スプライトレジスタに書き込まれた値はCYNTHIAによって解釈され，表示されます。

X68000のスプライトは以下のような仕様になっています。

画面上に最大128枚のスプライトを表示可能

のちに述べる特殊な方法により表示可能枚数を増やすこともできますが，基本的には128枚のスプライトを表示可能です。

16×16ドット16色のパターンを256枚まで定義可能

パターンデータは1枚につき128バイト（16色＝4ビットなので1バイトにつき2ドット，よって16×16=256ドットを表すのには128バイト必要）です。パターンデータ自体はSM.X（Oh!X1992年6月号付録ディスク掲載）やてぺ（激光電脳倶楽部7号掲載）などで作成可能です。

ここでスプライト≠パターンデータである点に注意してください。わかりやすい反例はゲームでよくある「同じパターンデータのキャラクターが複数登場する」シーンです。これはひとつのパターンデータを複数のスプライトに割り当てることにより実現しています。

16本のパレットブロックを使用可能

先ほどスプライトは16色と書きましたが，これは厳密には「パレット16色分」で，わかりやすくいうと「65536色中16色」のことです。この「65536色中16色」をまとめてパレットブロックと呼び，これがさらに16本用意されています。

パレットブロックを有効に利用した例としては「色違いキャラクター」が挙げられます（図1）。これは赤で書いたパターンデータと青で書いたパターンデータを用意しているわけではなく，パターンデータ自体は1種類で違うパレットブロックを指定することにより実現しています。

上下左右反転可能

読んで字のごとく。右を向いたキャラクターと左を向いたキャラクターがあった場合，これも左を向いたキャラクターのパターンデータを用意し，左右反転することにより実現しています。

優先順位指定可能

X68000には背景としてBGテキストという機構があり，これに対して背景より前に表示するか，後ろに表示するかを指定することができます。ただし今回はスプライトの説明ということもあり，背景に関しては割愛させていただきます。

以上からスプライトを表示するためには，

- パターンデータを定義
- パレットブロックを定義
- スプライトに対し，
  - どのパターンデータを使用するか？
  - どの座標に表示するか？
  - どのパレットブロックを使用するか？
  - 左右反転するか？
  - 優先順位は？

リスト1　sptest.c（IOCSによるスプライト表示）

```
---------- sptest.c
[1]#include <stdio.h>
[2]#include <stdlib.h>
[3]#include <string.h>
[4]#include <sys/iocs.h>
[5]
[6]#define PCG_NO       0        /* 定義するパターン番号 */
[7]#define PAL_BLOCK    1        /* 定義するパレットブロック */
[8]
[9]
[10]int main (int argc, char *argv[])
[11]{
[12]    FILE *fp;
[13]    char pcg_buf[128];
[14]    unsigned short pal_buf[16];
[15]    int i;
[16]
[17]    _iocs_crtmod (10);       /* 256x256ドット グラフィック画面 256色 2画面 */
[18]    _iocs_sp_init ();        /* スプライトの初期化 */
[19]    _iocs_sp_on ();
[20]
[21]    /* pcg_buf に一旦パターンデータを読み込む */
[22]    fp = fopen ("SPTEST.SP", "rb");
[23]    fread (pcg_buf, sizeof (char), 128, fp);
[24]    fclose (fp);
[25]    /* パターンデータを定義 */
[26]    _iocs_sp_defcg (PCG_NO, 1, pcg_buf);
[27]
[28]    /* pal_buf に一旦パレットデータを読み込む */
[29]    fp = fopen ("SPTEST.PAL", "rb");
[30]    fread (pal_buf, sizeof (unsigned short), 16, fp);
[31]    fclose (fp);
[32]    /* パレットデータを定義 */
[33]    {
[34]            unsigned short *p = pal_buf;
[35]            for (i = 0; i < 16; i++)
[36]                    _iocs_spalet (i, PAL_BLOCK, *p++);
[37]    }
[38]
[39]    _iocs_sp_regst (PCG_NO, 0, 128 + 16, 128 + 16, 0x0100, 3);
[40]/*  _iocs_sp_regst (PCG_NO+1, 0, 144+16, 144+16, 0x0100, 3); */
[41]    return (0);
[42]}
```

図2　スプライト128枚

図3　スプライト256枚

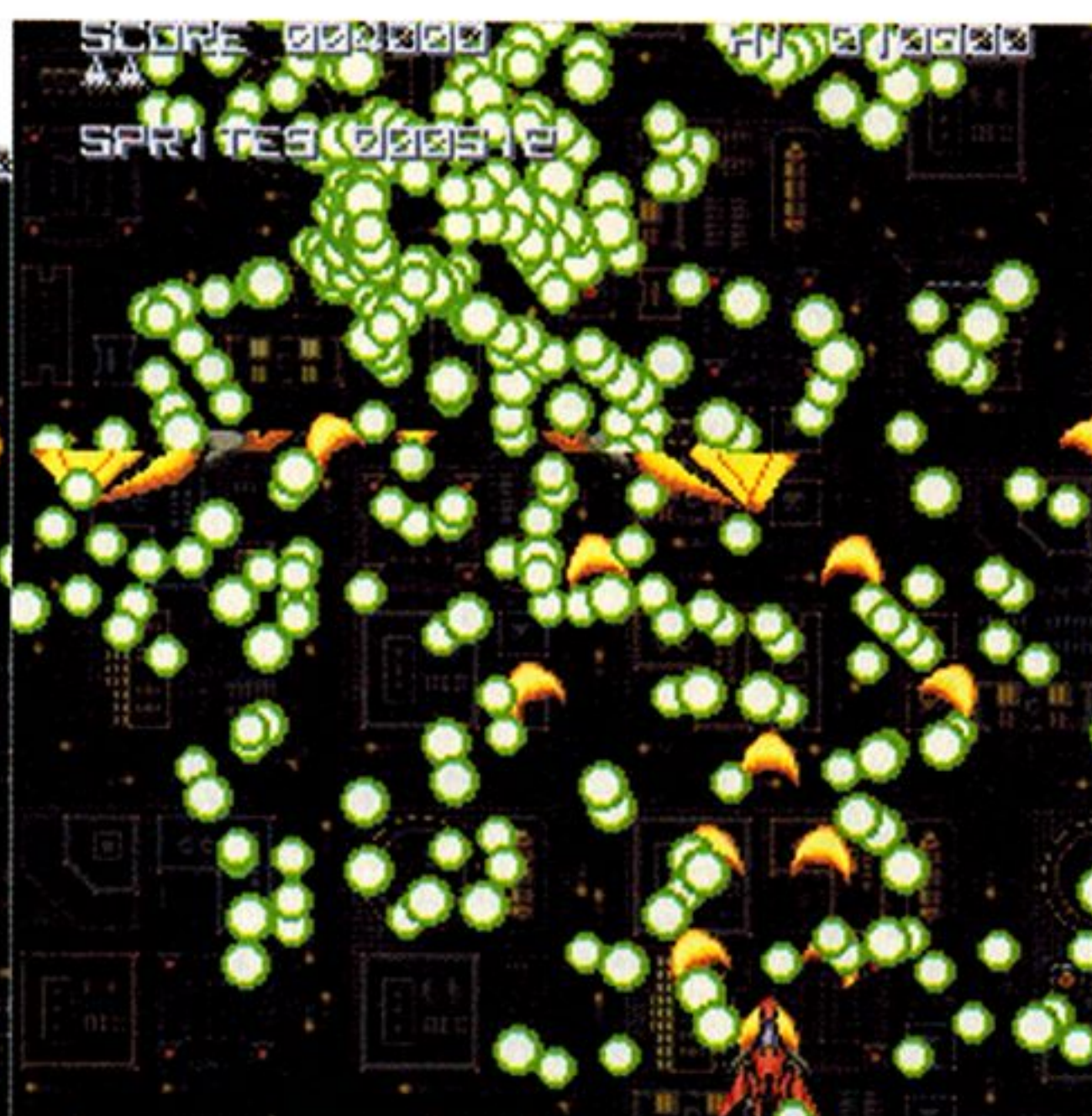



図4　スプライト512枚

といった情報が必要になることがわかります。

スプライトを使う

以上がわかったら実際にスプライトを表示してみます。開発環境はGCC＋LIBCで，スプライト表示にはIOCSコールを使用します。IOCSをC言語から使用するに関し必要なライブラリの仕様に関しては文献2を参照してください。

リスト1はスプライトを表示するサンプルです。順に解説しますと，17～19行は画面モードの設定およびスプライトの初期化です。21～26行はパターンデータSPTEST.SPをpcg_bufに読み込み，これを_iocs_sp_defcg()で定義しています。28～37行はパレットの定義で，SPTEST.PALをpal_bufに読み込んだあと，_iocs_spalet()で定義しています。パレットは1色につき2バイト(65536色)なのでサイズがunsigned shortになっています。unsigned shortが2バイトのコンパイラに依存した表記ですが，今回はX68000専用なので別にいいでしょう。ここではパレットブロック1番に16色分定義しています。

実際に表示する部分は39行です。_iocs_sp_regst()の引数は以下のとおりです。

```
int _iocs_sp_regst(int no, int mode, int x,
int y, int code, int prw)
```

no：定義するスプライト番号(0～127)
mode：最上位ビットが0ならば垂直同期を待って表示，1ならば待たずに表示
x,y：X座標，Y座標
code：パターンデータの番号(0～255)
prw：表示プライオリティ

スプライトの番号は0～127の中で指定します。これにより「X68000のスプライト表示枚数は最大128枚」ということができます。modeは垂直同期待ちの指定ですがここではとりあえず0を指定しておきます(垂直同期に関してはいずれまた……)。x，yは座標ですが，注意しなければならないのは「画面左上の座標が(16，16)」であるということです。ここで(0,0)を指定すると画面にスプライトが表示されず，悩みの種となるので要注意ポイントです。画面端の処理をしやすくするためにこのような仕様にしているのだと思うのですが……。prwはプライオリティで，ここでは3(スプライトはいちばん手前)を指定しています。

実行すると画面中央にXロゴが表示されます。コメントアウトしてある40行を有効にするともう1枚Xロゴを表示します。これは先ほど説明した「別々のスプライトに同じパターンデータを割り当てる」例です。

以上がIOCSコールを用いたスプライトの使用法です。

XSPの仕組み

スプライトを使用するにあたり便利な機能をパッケージングしたライブラリが公開されています。ファミベのよっしん氏作のXSPです(激光電脳倶楽部2号に掲載)。XSPは以下のような機能を備えてます。

スプライトを最大512枚表示可能

一般的にいうスプライトダブラー機能です(4倍だからクアドラプラーか)。

原理を解説します。まず，一般的にスプライトに限らず多くのCRTディスプレイは画面を1本(正確にはGBBで3本)の光線で描画していきます。光線は1/55秒の間に上から下へ，左から右へと画面を走っていきます。走っていく間に光線の明るさを加減すればドットの明るさを表現することができます。この走っていく光線を走査線＝ラスターと呼びます。

重要なのは画面が上から下へと描画されるということです。スプライトダブラーは，スプライトをY座標でソートし，ラスターが画面のいちばん上にいるときに画面上半分のスプライトを設定します。ラスターが画面の中央にきた頃には設定したスプライトはすべて表示が終了しますが，このときに画面下半分のスプライトを設定するとなにごともなかったように残りのスプライトが表示されます。このように上半分128枚＋下半分128枚＝256枚のスプライトを表示するのがスプライトダブラーです。

なお，ラスターが画面中央にきたことを検出するためには指定ラスター割り込みを使用します。XSPではこれを最大4回行い，1画面512枚のスプライトを実現しています。図2～4はスプライト枚数の比較表示です。図2はスプライト128枚，図3は256枚，図4は512枚の例です。図4は一部スプライトが消えているため正しい表示ではないのですが，それにしてもXSP，凄いですね。

この実装から想像できることは「スプライトレジスタはラスターが走っている瞬間に参照されている」という仕様です。参照が終わってしまえばその内容は用済みなわけですから，設定を変更してしまっても実害はありません。それどころか内容を再設定することにより再びスプライトを表示させてしまう，というアプローチは設定と参照のタイミングを見極めたハードウェアの制御方法として非常に興味深いものがあります。

大きなスプライトを表示可能

16×16ドット以上のキャラクターはスプライトを複数枚使って表示させる方法が一般的です。たとえば32×32ドットなら4枚，64×64ドットなら16枚のスプライトを使うわけですが，実際にプログラムしているときにこれらのことを考慮しなければならないのは面倒です。そこでXSPでは複数のスプライトをまとめて「複合スプライト」と呼ばれる単位で扱うことができます。複合スプライトの実体は複数のスプライトのパターン番号とオフセット座標からなる構造体です。

複合スプライトデータは表1のようなテキストフ

リスト2

```
---------- sptest2.c
[1]#include <stdio.h>
[2]#include <stdlib.h>
[3]#include <sys/iocs.h>
[4]#include "XSP2lib.H"
[5]
[6]#define PCG_MAX      32              /* パターンデータの個数 */
[7]#define FRM_MAX      128             /* フレームデータの個数 */
[8]#define REF_MAX      128             /* リファレンスデータの個数 */
[9]
[10]static char pcg_alt[PCG_MAX + 1];   /* ＰＣＧ配置管理テーブル */
[11]static char pcg_dat[PCG_MAX * 128]; /* ＰＣＧデータファイル読み込みバッファ */
[12]static XOBJ_REF_DAT ref_dat[REF_MAX];        /* リファレンスデータ */
[13]static XOBJ_FRM_DAT frm_dat[FRM_MAX];        /* フレームデータ */
[14]
[15]static unsigned short pal_dat[16];
[16]
[17]
[18]int main (int argc, char *argv[])
[19]{
[20]    FILE *fp;
[21]    int i, r;
[22]
[23]    _iocs_crtmod (10);       /* 256x256ドット グラフィック画面 256色 2画面 */
[24]    _iocs_sp_init ();        /* スプライトの初期化 */
[25]    _iocs_sp_on ();
[26]
[27]    /* pcg_dat にパターンデータを読み込む */
[28]    fp = fopen ("SPTEST2.XSP", "rb");
[29]    fread (pcg_dat, sizeof (char), PCG_MAX * 128, fp);
[30]    fclose (fp);
[31]
[32]    /* frm_dat にフレームデータを読み込む */
[33]    fp = fopen ("SPTEST2.FRM", "rb");
[34]    fread (frm_dat, sizeof (XOBJ_FRM_DAT), FRM_MAX, fp);
[35]    fclose (fp);
[36]
[37]    /* ref_dat にリファレンスデータを読み込む */
[38]    fp = fopen ("SPTEST2.REF", "rb");
[39]    r = fread (ref_dat, sizeof (XOBJ_REF_DAT), REF_MAX, fp);
[40]    fclose (fp);
[41]
[42]    /* REF_DAT[].ptr 補正 */
[43]    for (i = 0; i < r; i++)
[44]            (int) ref_dat[i].ptr += (int) frm_dat;
[45]
[46]
[47]    /* pal_buf に一旦パレットデータを読み込む */
[48]    fp = fopen ("SPTEST2.PAL", "rb");
[49]    fread (pal_dat, sizeof (unsigned short), 16, fp);
[50]    fclose (fp);
[51]    /* パレットデータを定義 */
[52]    {
[53]            unsigned short *p = pal_dat;
[54]            for (i = 0; i < 16; i++)
[55]                    _iocs_spalet (i, 1, *p++);
[56]    }
[57]
[58]    xsp_on ();
[59]    xsp_mode (3);
[60]    /* パターンデータを定義 */
[61]    xsp_pcgdat_set (pcg_dat, pcg_alt, sizeof (pcg_alt));
[62]    xsp_objdat_set (ref_dat);
[63]
[64]    xobj_set (128 + 16, 128 + 16, 0, 0x013f);
[65]    xsp_out ();
[66]    xsp_vsync (120);
[67]
[68]    xsp_off ();
[69]
[70]    _iocs_crtmod (16);
[71]    return (0);
[72]}
```

リスト3

```
---------- xsp2lib.h
[1]/* 複合スプライトのフレームデータ構造体 */
[2]typedef        struct  {
[3]     short   vx;                              /* 相対座標データ */
[4]     short   vy;                              /* 相対座標データ */
[5]     short   pt;                              /* スプライトパターンＮｏ. */
[6]     short   rv;                              /* 反転コード */
[7]} XOBJ_FRM_DAT ;
[8]
[9]/* 複合スプライトのリファレンスデータ構造体 */
[10]typedef       struct  {
[11]     short   num;                    /* 合成スプライト数 */
[12]     void    *ptr;                   /* 開始位置へのポインター */
[13]     short   unused;                 /* （未使用） */
[14]} XOBJ_REF_DAT ;
```

表1

```
PCG_FILE =          SPTEST2.SP          2          * 使用する PCG データ

XY_OFFSET              =        $0000     $0000    * 座標のオフセット
PT_OFFSET              =        $0000              * PCGパターンナンバーのオフセット
OBJ_RV                 =        $0000              * 全体の反転コード

****************************** 複合スプライトパターン  0 ****
No.            =        0           * 複合スプライトのパターンナンバー
*          X座標   Y座標          パターン番号          反転コード
           -16      -16            0                   $0000
            0       -16            0                   $4000
           -16       0             1                   $0000
            0        0             1                   $4000

                                 * 反転コードは
                                 * $0000 = 反転なし
                                 * $4000 = 左右反転
                                 * $8000 = 上下反転
                                 * $c000 = 上下左右反転
```

図5　スプライトを組みあわせてキャラクターを作る

ァイル（拡張子.OBJ）で与え，これをXSPに同梱のCVOBJ.Xで変換して使用します。図5の例は，

座標(-16，-16)にパターン0を反転なし
座標(0，-16)にパターン0を左右反転
座標(-16，0)にパターン1を反転なし
座標(0，0)にパターン1を左右反転

で表示する複合スプライトです。このとき，複合スプライトの座標として(128，128)を指定すると，

座標(112，112)にパターン0を反転なし
座標(128，112)にパターン0を左右反転
座標(112，128)にパターン1を反転なし
座標(128，128)にパターン1を左右反転

の4枚のスプライトが表示されます。

CVOBJ.Xで出力されるデータは拡張子.XSP/.FRM/.REFの3ファイルです。.XSPはスプライトデータそのものですが，未使用パターンは削除され，上下左右反転で表示できるパターンに関してはひとつにまとめられます。.FRMはフレームデータ構造体，.REFはリファレンスデータ構造体そのものです。内容に関してはリスト3をご覧ください。

最大32768枚のパターンデータを定義可能

パターンデータは前述のとおり256枚まで定義可能ですが，これを32768枚に拡張します。未定義のパターンが出現した場合，パターンをメインメモリからパターンデータに転送することによりこれを実現しています。

XSPを使う

では実際にXSPを使用してみます。リスト2はXSPを使用したサンプルです。実行すると画面中央に戦闘機らしきものを表示して終了します。表2にXSPの簡易関数リファレンスを挙げておくのであわせてご覧ください。

リスト2，6～8行目は各種定数の設定です。サンプルということもあり，多少大きめの数値を設定しています。10～13行は各種ワークエリアの設定です。27～40行でスプライトデータをpcg_datに，フレームデータをfrm_dataに，リファレンスデータをref_datに読み込んでいます。42～44行はREF_DAT[].ptrの補正です。REF_DAT[].ptrは参照するフレームデータへのポインタを代入するのですが，アドレスは実行時まで確定できないためここで代入しています。61～62行で先ほどのpcg_dat，ref_datを設定し，xobj_set()で複合スプライトを設定しています。

図6　爆発パターンの設定

xobj_set()関数自体は内部のワークエリアに値を設定するのみで，実際の表示は次の行のxsp_out()で行っています。

以上が簡単なXSPの使用法です。

複合スプライトの変則的な使用法

上記のように複合スプライトは表示するパターン番号と座標のセットにより構成され，それらは.OBJファイルで定義します。先ほどの例では座標を16ドット単位で指定し，矩形にスプライトを敷き詰めてみましたが，別にそうでなくてはならないという理由はありません。また，.OBJファイルを手動で作成しなければならないという理由もありません。そこで今回の主題である物理法則に基づいたスプライトの動きになだれ込みます。

物理法則の事前処理

作成するサンプルは図6のような爆発パターンです。これは6パターンからなる小さな爆発パターンを組み合わせて大きな爆発パターンを表現しています。小さな爆発パターンは中央に出現し，アニメーションしつつ外側に拡散していきます。アニメーションはパターン番号を1ずつ増やしていくだけで実現できますが，外側に拡散＝移動は単純そうに見えて，実は結構難しかったりします。単純に直線運動をしていくだけではリアルに見えないので加速度を使用したいところですが，座標値が整数だと精度が足らないため綺麗に拡散していきません。浮動小数点演算を使うという方法もあるにはあるのですが，いかんせん遅いためゲーム中には使いたくありません。固定小数点を使うという方法もあるのですが……。

そこでここでは違うアプローチをとってみます。「複数の小さな爆発パターンで構成された爆発パターンを複合スプライトとして定義してしまう」のです。こうすれば表示時に計算は不要なので高速に処理できます。複合スプライトを使用するため，その関連データ(.FRMや.REF)をメモリ上に用意しなければならないため，若干メモリを消費しますがそれは微々たるものです。

さて，これを実現するための具体的な方法は「複合スプライトを定義する.OBJファイルを自動

表2　XSP簡易リファレンスマニュアル

void xsp_on();
XSPシステムの初期化を行う。

void xsp_off();
XSPシステムの解放を行う。

void xsp_pcgdat_set(void *pcg_dat, char *pcg_alt, short alt_size);
引数：void *pcg_dat：パターンデータへのポインタ
　　　char *pcg_alt：パターン配置管理テーブルへのポインタ
　　　short alt_size：パターン配置管理テーブルのサイズ
パターンデータをパターン配置管理テーブルに登録する。テーブルはパターンデータの枚数＋1バイト必要。

void xsp_mode(short mode_no);
引数：short mode_no：1＝128枚ちらつかせ最大384枚モード
　　　　　　　　　　2＝最大512枚モード（デフォルト）
　　　　　　　　　　3＝最大512枚優先度破綻軽減モード
XSPの動作モードを指定する。

short xsp_vsync(short n);
引数：short n：垂直帰線期間数
指定された垂直同期期間待つ。

short xsp_set(short x, short y, short pt, short info);
引数：short x　：スプライトX座標
　　　short y　：スプライトY座標
　　　short pt　：スプライトパターン番号（0～0x7fff）
　　　short info：反転コード・色・表示優先度を表すデータ
スプライトを登録する。本関数はXSP内部ワークへの登録を行うだけであり，実際に表示するためには xsp_out() を実行する必要がある。

short xobj_set(short x, short y, short pt, short info);
引数：short x　：複合スプライトX座標
　　　short y　：複合スプライトY座標
　　　short pt　：複合スプライトパターン番号（0～0x0fff）
　　　short info：反転コード・色・表示優先度を表すデータ
複合スプライトを登録する。本関数はXSP内部ワークへの登録を行うだけであり，実際に表示するためには xsp_out() を実行する必要がある。

short xsp_out();
スプライトを実際に画面に表示する。

リスト4

```
---------- main.c
[1]#include <stdio.h>
[2]#include <stdlib.h>
[3]#include <XSP2lib.H>
[4]#include <sys/iocs.h>
[5]#include "sprite.h"
[6]#include "fxsp2lib.h"
[7]
[8]#define      PCG_MAX 32                /* スプライトP
CGパターン最大使用数 */
[9]static char pcg_alt[PCG_MAX + 1];     /* PCG配置管
理テーブル */
[10]static char pcg_dat[PCG_MAX * 128]; /* PCGデータ
ファイル読み込みバッファ */
[11]static unsigned short pal_dat[16];
[12]
[13]/* 爆発パターンを出現させる個数 */
[14]static int seq_data[]=
[15]{5, 8, 6, 3, 2, 1};
[16]
[17]
[18]void Init (void)
[19]{
[20]    FILE *fp;
[21]    int i;
[22]    unsigned short *p = pal_dat;
[23]
[24]    _iocs_crtmod (10);       /* 256x256ドット グラフ
ィック画面 256色 2画面 */
[25]    _iocs_sp_init ();        /* スプライトの初期化 */
[26]    _iocs_sp_on ();
[27]
[28]    /* pcg_dat にパターンデータを読み込む */
[29]    fp = fopen ("EXPL.SP", "rb");
[30]    fread (pcg_dat, sizeof (char), PCG_MAX * 128,
fp);
[31]    fclose (fp);
[32]
[33]    /* pal_buf に一旦パレットデータを読み込む */
[34]    fp = fopen ("EXPL.PAL", "rb");
[35]    fread (pal_dat, sizeof (unsigned short), 16,
fp);
[36]    fclose (fp);
[37]    /* パレットデータを定義 */
[38]    for (i = 0; i < 16; i++)
[39]            _iocs_spalet (i, 1, *p++);
[40]
[41]    SpriteInit0 ();
[42]
[43]    xsp_on ();
[44]    fxsp_on ();
[45]    xsp_mode (3);
[46]    /* パターンデータを定義 */
[47]      xsp_pcgdat_set (pcg_dat, pcg_alt, sizeof
(pcg_alt));
[48]}
[49]
[50]
[51]int Tini (void)
[52]{
[53]    xsp_off ();
[54]    fxsp_off ();
[55]    _iocs_crtmod (16);
[56]
[57]    return (0);
[58]}
[59]
[60]
[61]void Loop (void)
[62]{
[63]    int seq_counter = 0;
[64]
[65]    /* スプライトが表示されている間ループ */
[66]    do {
[67]        /* 爆発パターンを出現させるか? */
[68]         if (seq_counter < (sizeof (seq_data) /
sizeof (int))) {
[69]                                           int  i  =
seq_data[seq_counter];
[70]        while (i--) {
[71]                                 double x, y, vx, vy;
[72]         x = drand ()* 16.0 - 8.0;          /* 座標
*/
[73]        y = drand ()* 16.0 - 8.0;
[74]        vx = x / 1.7;   /* 速度 */
[75]        vy = y / 1.7;
[76]         SpriteInit (x, y, vx, vy, (char) (rand
()% 6 + 1));
[77]        }
[78]        }
[79]        seq_counter++;
[80]
[81]        SpriteMove ();
[82]        fxsp_out ();
[83]    } while (xsp_out ());
[84]
[85]    xsp_vsync (120);
[86]}
[87]
[88]
[89]
[90]int main (int argc, char *argv[])
[91]{
[92]    Init ();
[93]    Loop ();
[94]    Tini ();
[95]
[96]    return (0);
[97]}
```

リスト5

```
---------- sprite.c
[1]#include <stdlib.h>
[2]#include <stdio.h>
[3]#include <XSP2lib.H>
[4]#include "sprite.h"
[5]#include "fxsp2lib.h"
[6]
[7]#define SPRITE_MAX   32        /* 最大数 */
[8]
[9]typedef struct _sprite {
[10]    signed short sx, sy;     /* 座標 */
[11]    short pt;                /* スプライトパターンNo. */
[12]    short info;              /* 反転コード・色・優先度を表わすデータ */
[13]    double x, y;             /* 座標 */
[14]    double vx, vy;           /* 速度 */
[15]    char cyc;                /* アニメーションサイクル初期値 */
[16]    char cyc1;               /* 現在のアニメーションサイクル */
[17]    struct _sprite *next;    /* 次の構造体へのポインタ */
[18]} SPRITE;
[19]
[20]
[21]static SPRITE sprite[SPRITE_MAX];   /* ワーク */
[22]static SPRITE *sprite_top, *sprite_null_top;
[23]
[24]
[25]/* 起動時に1度だけ呼ばれる */
[26]int SpriteInit0 (void)
[27]{
[28]    int i;
[29]
[30]    sprite_top = NULL;
[31]    sprite_null_top = sprite;
[32]    for (i = 0; i < SPRITE_MAX; i++)
[33]            sprite[i].next = &sprite[i + 1];
[34]
[35]    sprite[SPRITE_MAX - 1].next = NULL;
[36]
[37]    return (0);
[38]}
[39]
[40]
[41]/* 爆発パターンが出現する度に呼ばれる */
[42]void SpriteInit (double x, double y, double vx, double vy, char cyc)
[43]{
[44]    if (sprite_null_top != NULL) {
[45]            SPRITE *p;
[46]
[47]            p = sprite_null_top;
[48]            sprite_null_top = p->next;
[49]            p->next = sprite_top;
[50]            sprite_top = p;
[51]
[52]            p->x = x;
[53]            p->y = y;
[54]            p->pt = 0;
[55]            p->info = ((rand ()% 15) << 12) | 0x013f;
[56]            p->vx = vx;
[57]            p->vy = vy;
[58]            p->cyc = cyc;
[59]            p->cyc1 = cyc;
[60]    }
[61]}
[62]
[63]
[64]/* 垂直同期ごとに呼ばれる */
[65]void SpriteMove (void)
[66]{
[67]    SPRITE *p, *q;
[68]
[69]    p = sprite_top;
[70]    q = NULL;
[71]    while (p != NULL) {
[72]            char erase_flag = 0;    /* 消去するか? */
[73]
[74]        /* 速度を足す */
[75]            p->sx = (p->x += p->vx);
[76]            p->sy = (p->y += p->vy);
[77]
[78]            p->vx -= p->vx * 0.2;   /* 空気抵抗 */
[79]            p->vy -= p->vy * 0.2;
[80]
[81]            if (--p->cyc1 <= 0) {
[82]                    p->cyc1 = p->cyc;
[83]                    if (++p->pt > 5)
[84]                            erase_flag = !0;
[85]            }
[86]            if (erase_flag) {
[87]                    if (q == NULL) {        /* リストの一番最初を削除 */
[88]                            sprite_top = p->next;
[89]                            p->next = sprite_null_top;
[90]                            sprite_null_top = p;
[91]                            q = NULL;
[92]                            p = sprite_top;
[93]                    } else {
[94]                            q->next = p->next;
[95]                            p->next = sprite_null_top;
[96]                            sprite_null_top = p;
[97]                            p = q->next;
[98]                    }
[99]            } else {
[100]                   p->sx += (128 + 16);    /* 表示用に補正 */
[101]                   p->sy += (128 + 16);
[102]                   p->sx -= 8;
[103]                   p->sy -= 8;
[104]                   xsp_set_st (p);
[105]                   p->sx -= (128 + 16);
[106]                   p->sy -= (128 + 16);
[107]                   fxsp_set_st (p);
[108]                   p->sx += 8;
[109]                   p->sy += 8;
[110]                   q = p;
[111]                   p = p->next;
[112]           }
[113]   }
[114]}
```

生成する」ことです。.OBJファイル作成時は速度が要求されないため，思う存分浮動小数点数が使えます。

では実際の例を見てみましょう。リスト4～6が図6のような複合スプライトの.OBJファイルを作るためのプログラムEXPL.Xです。画面に実際にスプライトを表示しつつ，パターン番号とその座標をEXPL.OBJに書き出します。要するにXSPのライブラリに渡される引数を横取りしてファイルに書き出せばよいので，最初はXSP互換のダミーのライブラリに差し替えることも考えましたが，それも面倒なのでXSPと同じ形式で引数を受け取り，ファイルに出力するfxsp2libを作成しました。リスト4，43～44行がそれに該当します(fxspのfはファイル)。

以下，XSPを呼び出すところでfxsp2libを呼び出す(関数名の頭にfをつける)だけで並行してファイルに出力されます。61行からがメインループで爆発パターンが画面上に出ている間ループし，なくなると終了します。68～69行で爆発パターンを出現させるかどうか判定します。配列seq_data[]にはパターンを出現させる個数が14～15行で定義されています。72～75行はパターンの初期座標および初期速度です。初期速度は初期座標に比例するようにしてあります。厳密に行うならば火薬の爆発力から初期速度を算出すべきなのでしょうが，今回はそれらしく見えるようにということでこのようになっています。

リスト5，25行からは初期化ルーチンでワークエリアをリスト構造でつないでいます。ワークエリアをリスト構造で表現することにより高速かつ動的に挿入/削除ができるためゲームプログラミングでは必須のテクニックです。もっとも，ここでは速度は要求されないため別にリストを使うまでもないのですが，教育的な意味もあり，残しておきました。

41行からは爆発パターンが1個出現するたびに呼ばれる関数です。初期座標(x，y)と初期速度(vx，vy)と「何フレームごとにパターンを進めるか」を指定するcycを引数として受け取ります。なおフレームとはX68000の場合1/55秒のことを指します(31kHz高解像度モード時)。爆発パターンは上下左右反転しても使えるため，55行で乱数で反転を指定しています。

64行からは本来は垂直同期ごとに呼ばれるべきルーチンなのですが，ここではファイルの書き出しを並行して行っているため，高速機以外では若干遅れが生じます。が，前述のとおりファイルへの書き出しが目的のプログラムですので実害はありません。74～76行で速度を足しています。p->sxはshortに変換した座標です。XSPが引数としてshortの値を受け取るため変換しています。78～79行は空気抵抗を表現しています。物理法則では流体中を移動する物体は速度vに比例する抵抗力Fを受けます。これを式で表すと，

$$F = -Kv \quad (K\text{は定数})$$

となり，これを粘性抵抗力といいます。爆発パターンは空気中を移動すると仮定し，粘性抵抗力を受けることにより減速していく，ということを表現すると先ほどのようなプログラムが導き出せる

わけです。

問題は定数Kなのですが，空気の粘性抵抗係数や物体の質量や形状が不明なため，ここでは適当な数値を使用しています(あちゃー)。それでもそれなりにそれっぽくは見えているようです。ためしにこの2行をコメントアウトするとなんとも味気ない爆発になります。

100行以降は表示を補正するための小細工です。画面中央に表示するために128＋16を加算しているほか，16×16ドットのスプライトの(8,8)を中心とみなすため，8を減算しています。以上で算出したスプライトを104行で画面に表示し，107行でファイルに出力しています。

物理法則の応用

以上が粘性抵抗力の応用例です。ほかに考えられる用途をいくつか挙げてみましょう。

破片の飛び散り

敵キャラクターを爆破したときに破片が飛び散る表現，これも先ほどの例と同様にできます。破片の場合，大きな破片は空気抵抗が大きいことから粘性抵抗係数Kを大きめに，小さな破片の場合はKを小さめに取るとリアルに見えるでしょう。

パーツの合体

ボスキャラ本体が登場した後，部品パーツがどこからともなく飛んできて合体，というような表現の場合，それぞれが合体の衝撃で振動するとリアルに見えるでしょう。振動の要因は合体により運動量が伝わることによるものですから，これは力積で求められます。式で表すと，

$$MV + mv = 0$$

です。この式からいえることは合体後，質量が大きな本体は小さな速度で，質量が小さなパーツは大きな速度で動くようになるということです。速度はゲーム中で実際に使われる速度を使用するとしても，問題はパーツの重量をどうやって求めるか，です。

式を見ると重要なのはMとmの比率ですから，スプライトだったら透明でないドットの総面積の比率で近似するという方法があります。3Dだったら物体の形状データから体積を求め，それに材質の密度を掛ければ精確な数値が算出できるでしょう。

さて，ここで扱うのは単なる衝突ではなく合体ですから，しばらくすると逆方向に衝撃を受けることになります。さらに接合部は摩擦を受け，運動エネルギーが熱に変換されることから次第に速度が減衰し，0になります。ここまで精確にシミュレートするならば，接合部の摩擦係数が必要になります。そこまで厳密な計算が必要でなければこれを摩擦つきの単振動に近似して計算するという手もあります。

最後に

スプライトの講座に見せかけて実は物理法則シミュレーションへの誘いという内容でしたが，いかがでしょうか。物理法則はとても奥が深い内容で，とっつきにくい分野ではあるのですが，別に最新機種で3Dを使わなくても，なおかつリアルタイムに演算しなくてもこのように応用が利く手法なのです。

しかし今回悔やまれるのは各定数値･係数がわからなかったこと。物理法則はシミュレートできても肝心のこれらの数値が「画面で見て，経験的に設定した値」であるため，あくまでも「シミュレーションもどき」レベルに留まっています。まあ別にシミュレーションがすべてではありませんし，「それっぽく見える」だけでも十分な用途が多いので決して無駄ではないでしょう。

リスト6

```
---------- fxsp2lib.c
[1]/* fxsp2lib.c */
[2]
[3]#include <stdlib.h>
[4]#include <stdio.h>
[5]#include "fxsp2lib.h"
[6]
[7]
[8]typedef struct {
[9]     signed short vx;        /* 相対座標データ */
[10]    signed short vy;        /* 相対座標データ */
[11]    unsigned short pt;      /* スプライトパターンNo.  */
[12]    unsigned short rv;      /* 反転コード */
[13]} SP_WORK;
[14]
[15]static FILE *fp;
[16]static int pattern_no;
[17]static int sprite_no;
[18]
[19]
[20]
[21]short fxsp_on (void)
[22]{
[23]    if ((fp = fopen ("expl.obj", "wb")) == NULL) {
[24]            printf ("ファイルを書き込めません\n");
[25]            return (-1);
[26]    }
[27]    fprintf (fp, "\nPCG_FILE        =       EXPL.SP 256     * 使用する PCG データ\n"
[28]            "\n"
[29]            "XY_OFFSET      =       $0000   $0000   * 座標のオフセット\n"
[30]            "PT_OFFSET      =       $0000           * PCGパターンナンバーのオフセット\n"
[31]            "OBJ_RV                 =       $0000           * 全体の反転コード\n"
[32]            "\n");
[33]    pattern_no = 0;
[34]    sprite_no = 0;
[35]    return (0);
[36]}
[37]
[38]
[39]
[40]static void write_pos (signed short pos)
[41]{
[42]    signed short p = pos;
[43]
[44]    if (p < 0)
[45]            fprintf (fp, "-$%04hx   ", -p);
[46]    else
[47]            fprintf (fp, " $%04hx   ", p);
[48]
[49]    return;
[50]}
[51]
[52]
[53]
[54]/* xsp_set_st と置き換えて使うと引き数の構造体を .obj ファイルに書き出す */
[55]short fxsp_set_st (void *sp_work0)
[56]{
[57]    SP_WORK *sp_work = (SP_WORK *) sp_work0;
[58]
[59]    if (!sprite_no) {
[60]            fprintf (fp, "******************************** 複合スプライトパターン %3d ****\n\n", pattern_no);
[61]            if (pattern_no == 0) {
[62]                    fprintf (fp, "No.       =       0       * 複合スプライトのパターンナンバー\n");
[63]            } else {
[64]                    fprintf (fp, "No.       =       NEXT    * 複合スプライトのパターンナンバー\n");
[65]            }
[66]            fprintf (fp, "                          * 左から順に、X Y PT RV （以下同様）\n");
[67]    }
[68]    fprintf (fp, "  ");
[69]    write_pos (sp_work->vx);
[70]    write_pos (sp_work->vy);
[71]
[72]    fprintf (fp, "$%04x     ", sp_work->pt);
[73]    fprintf (fp, "$%04x\n", sp_work->rv);
[74]    sprite_no++;
[75]
[76]    return (0);
[77]}
[78]
[79]
[80]
[81]short fxsp_out (void)
[82]{
[83]    short s = sprite_no;
[84]    fprintf (fp, "\n\n");
[85]    pattern_no++;
[86]    sprite_no = 0;
[87]
[88]    return (s);
[89]}
[90]
[91]
[92]
[93]short fxsp_off (void)
[94]{
[95]    if (fp != NULL)
[96]            fclose (fp);
[97]    fp = NULL;
[98]
[99]    return (0);
[100]}
```

参考文献

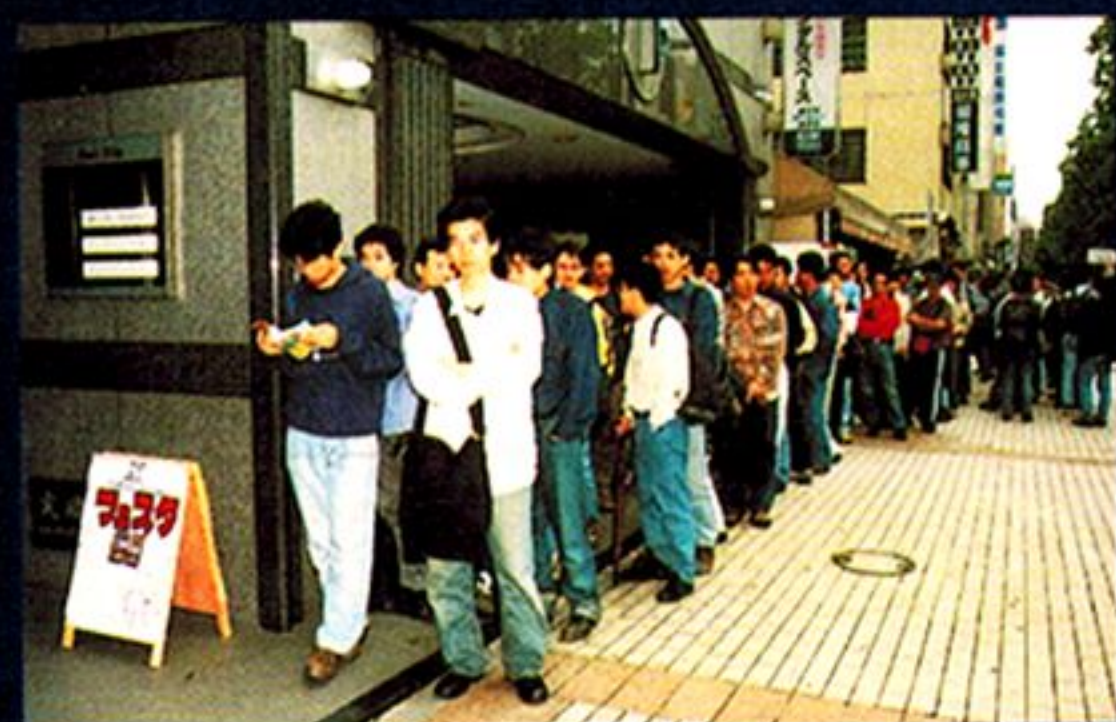
1) Inside X68000　桑野雅彦著(ソフトバンク刊)

2) X68k Programming Series#2 X680x0 libc　村上敬一郎　大西恵司　荻野祐二　共著

3) XSP Ver2.00 マニュアル　ファミベのよっしん

フェスタ68レポート

column

どうも「フェスタ・68」へ足を運んでいただいた皆さん，ありがとうございました！お蔭様で予想をはるかに上まわり，開会2時間前から長蛇の列ができるほどの盛況となりまして，スタッフ一同感激しております！ え～前回は実験的な要素が大きかったので，安くて場所が秋葉原から近くてそこそこの広さで構わないと，安易に考えてあの会場に決めたために，参加していただいた皆さんに多大な御迷惑をおかけする混雑となりました。

開場前の長蛇の列

一応宣伝と動員数の目安を兼ねて，チケットぴあを利用しましたが，最後の最後まで売り上げは30枚とか50枚程度だったので，てっきり200人もきてもらえれば運がいいほうだと思っておりました。このことから予想以上にチケットぴあ経由の販売には，馴染みがなかったようです。当日の混雑を見たあとからでも，もう1フロア貸りておけば，少しは回避できたのではないかと，いまさらながら反省しております。

中古販売されたX68030の山！壮観

しかし，今回のことでまだまだX68000の伝説は続くことも，よくわかりましたのでこれからもこういったかたちでの支援ができたらと，考えております。

え～挨拶はこのくらいにして……当日は本当に凄いというか激しい動員数でしたぁ～開場してから閉会するまで人が絶えることがなかったですからねぇ～。まぁその分，いろいろな問題がありましたから，よいことばかりじゃないので次回はその辺りの改善も進めていこうと思いますが，主催者側ではどうすることもできない問題も多々ありますので，ここへちょっと書いてみますね。

まずは各サークルさんの商品が開会からわずかな時間で売り切れ状態になってしまった問題ですが，どのサークルさんも小人数で運営されていますからそれ程多く商品が用意できないと思いますが，当日の動員数が予測できれば少しは緩和されるんじゃないかとも思うので，事前に販売枚数を知ることができる前売り券をできるだけご利用ください。

ゲーム大会もなかなかの盛況

それからこれはしかたないことなのかもしれませんが，ひとつのサークルさんのスペースにあまり大勢の方が集まるのは，通路をふさいだり隣接のサークルさんに迷惑がかかりますから，どうぞ理性を持った行動をお願いいたします。

X68000にK6-2マザーを入れてしまった人

また，事前に混雑が予想されるサークルさんは，お手数ですが前もって整理券などを準備するなどしていただいて，少しでも当日の混雑緩和に御協力くださいな。

大体こんなところが特に問題となっていたと思いますので，今こそユーザーのパワーを結集し，誰もが気持ちよく参加できる，よりよいイベントを目指しましょう！

変わり種としてはあまりのお客の数に，完売したサークルさんの一部は現地でCD-RやMOを焼いていたのが，面白かったですねぇ～。場所が秋葉原ということもあって，気軽にメディアを買いに行けるのはよかったですね。

その中身

辛かったのは途中暑くてクーラーを最強にしたのですが，なかなかきかなかったり，ダクトが結露して真下のサークルさんが犠牲になったりとハプンニング続出でしたが，最後には楽しかった，また参加したいと温かいお言葉をたくさんいただけてその日の疲れも，一気にふっ飛びました。

第2回「フェスタ・68」開催のお知らせ！

1999年10月11日（月曜・振り替え休日）
開催時間：10時～15時（サークル入・退場時間は，9時～16時）
会場場所：秋葉原駅とお茶ノ水駅の間に位置する，『損保会館』2Fです。

一般参加は，できるだけ当日の混雑をさけるために，全国のチケットぴあから販売される前売り（300円）を買ってください。当日券は500円とします。コンビニや本屋など意外に身近な所で，扱ってることもありますのでお手数ですが皆さんどうぞよろしくお願いします。その際に必要なPコードは<500-502>と，なっております。

サークル参加は前回と同様で，180cm×45cmの机半分を1スペースとします。1スペース3000円で，1サークル2スペース（6000円）までとさせていただきます。お代は開催当日受付へお支払いいただく，完全後払い制でキャンセル料などのリスクが一切ありませんので，少しでも参加の御意志をお持ちのサークルさんはぜひ！ぜひ！お早めに参加表明くださいませ。また今回も電源が使用できますので，デモにゲーム大会にとご利用ください。

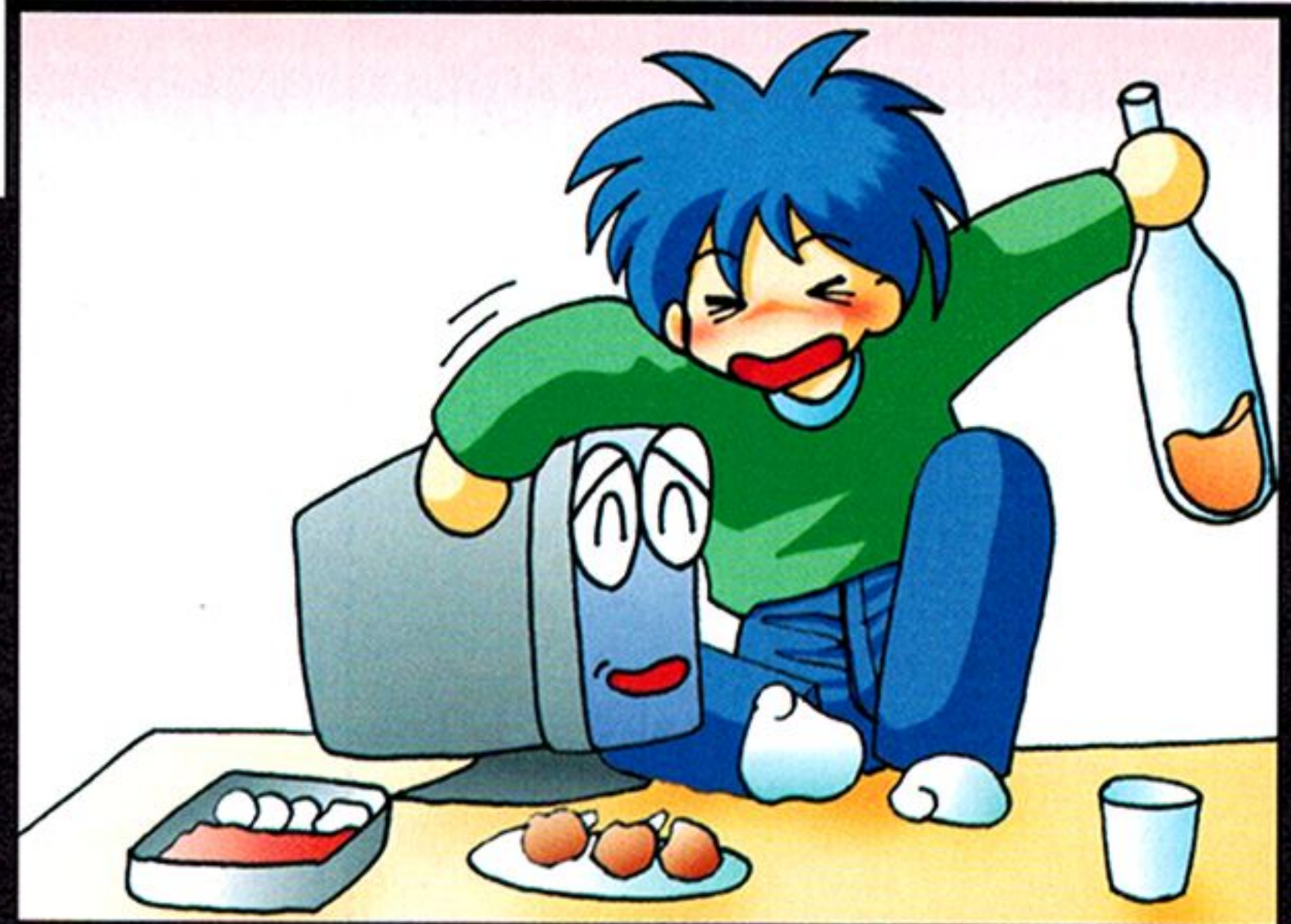
会場には駐車スペースもありますが，結構高いので近所に点在する安価な駐車場をおすすめします。なお，荷物の積みおろしは30分まで無料で行えます。そんなわけで，現在参加サークルを大大的に募集しております！ 詳しい情報/サークル参加は以下のHPでお願いします。

http://www.geocities.co.jp/SiliconValley-PaloAlto/4247/

また満開ネット（03-3985-6204）のデータバンクボード（¥25）にも，サークル参加表明のフォームを用意しておりますので，こちらからもお受けいたしております。

（しゃかんきょりたもつ）

投稿募集

We want you!

明日は自分の手で作れ

ご覧のとおり，Oh!Xは復刊しましたとはいえ，まだまだ至らない点も多く，ジャンルにもかなり偏りが生じています。もっともっと多彩な記事を掲載していくのが「よりよいOh!X」を模索していくうえでも必要と考えています。現状のスタッフもそれなりに強力な面々が並んでいると自負していますが，なにぶん兼業スタッフが多く，どうしても十分な対応が望めません。新しいOh!X構成していくにはまだまだ人手が足りません。新しい戦力を広く募集しています。

●Oh!Xスタッフ募集

前回のスタッフ募集に応募していただいた方すみません。連絡が行き届かずご迷惑をおかけしております。かなり多くの方からご協力申し出のお手紙をいただいたのですが，残念ながらこちらの手不足で管理しきれておりません。

現状では，連絡網の確立やファイルの受け渡しなどを考えるとインターネットアクセス環境必須，またはソフトバンクパブリッシング本社（東京都中央区日本橋箱崎町）まで直接来社可能なことというのが絶対条件になりそうです。なにぶん変則的な体制で編集作業を進めておりますので，ご理解ください。

●それでもスタッフ募集

引き続き各分野で人材を募集しています。あらゆる分野でパソコンおよびさまざまなモノに対してあくなき探求心を持った方を募集しております。とりあえず，「なにかをやりたい」というのが最大のポイントで，さらにいえばより具体的に「～ができる」とか「～がやりたい」と具体的に書いていただけるとこちらも把握しやすいです。ただ，上記のように，インターネット接続環境はほぼ必須と考えてください。

募集するのは基本的には執筆スタッフで，なんらかのテーマに従って記事を書いていただきます。原稿なんか書いたことがないという人でも，経験上，たいていは大丈夫です。プログラミング，ハードウェア制作，ゲームレビュー，ソフトウェアレビュー，その他技術解説などなど，その他，どんな記事でもかまいません。

記事を書くのは自信がないという方ではプログラム制作やビジュアル制作（イラストなど）に絞った仕事をしていただくこともあります。どのような系統のものをやりたいのかなどを明記してご応募ください。

なお，ジャンルについては，現状の記事内容はあまり参考になりません。新しい分野をどんどん開拓していくつもりでいてください。皆様のチャレンジングな企画をお待ちしています。

●投稿募集

スタッフとして参加するのは無理という方でも，なんらかのかたちで協力していただける方を広く募集しています。記事投稿，プログラム投稿なども随時受け付けています。ゲームレビューや単独記事でもかまいません。

プログラム投稿の場合は，機種などはといませんが，対応機種（開発機種），OSのバージョン，ライブラリなどの開発環境のバージョンと実行環境の詳細なども明記するようにお願いします。

ゲームレビュー記事なども募集しています。2号分のレビューを見ていただいた方はなんとなく察しているかもしれませんが，ゲームレビュー記事ではゲームの新旧は問いません（画面写真が撮れる/入手できるくらいのものであることが望ましいが）。もちろん，機種も問いませんし，以前に誌面上で扱われているタイトルでもかまいません。それなりに遊び込んだ作品を，通り一遍でない評価で記事にしてください。複数のタイトルで記事を構成することも可能です。

この辺はちょっと人手不足気味ですので，皆さんの積極的な参加をお待ちしております。

●Oh!X LIVE in '99

音楽データの場合は，多少注意点が異なりますので下記の条項をよく確認してください。曲データのジャンルや種別は特に問いません。ただし著作権の関係で掲載できないデータというものも存在します。これは個別に調査しないと判定できないものが多いので，あらかじめご了承ください。

それでは投稿時の注意です。

まず，使用する機材の構成を明記し，オリジナル曲か原作つきの曲か，原作つきの曲の場合，作曲者は誰かなどの情報を明記して，演奏データを送ってください。その際にカセットテープ，CD-Rなどで実際の演奏を録音したものを添付してください（MD，DATなどはご遠慮ください）。ほぼ例外的にX68000内蔵音源のみの場合は特に必要ではありませんが，MIDI使用の場合は録音物の添付をお願いします。

特にゲームミュージックの場合はなにのどの曲かというのを詳しくお願いします。

データの形態はMMLではなく，リアルタイム入力や，シーケンサを使った曲などでもかまいません。使用ツール名などを明記して，SMFだけでなくソースデータファイルも付属してください。

機種固有音源やMIDI使用曲などではデータと一緒に必ず録音テープを添えてください。こちらで同じ環境での再生ができるとは限りません。

いずれの場合も，

〒103-8501
東京都中央区日本橋箱崎町24-1
ソフトバンクパブリッシング（株）
DOS/V magazine編集部
Oh!X制作実行委員会

までお送りください。なお，内容に応じて，

プログラム投稿係
スタッフ募集係
Oh!X LIVE in 係

など，適当に宛先係名を追加しておいてください（厳密な区別はありません）。

または，e-mailで aueki@softbank.co.jp までご連絡ください。

なお，投稿いただいた作品の返却はいたしておりませんので，あらかじめご了承ください。

コンピュータアーキテクチャ

その直感的アプローチ ❶

MMUの機能を探る

中森 章 *Nakamori Akira*

革新的なアーキテクチャを実現するために「定量的な分析」ではなく，あえて「直感をもって行う」アプローチを取ることも必要だと中森氏は強調する。ここではコンピュータを構成するさまざまな基本技術について，そのアーキテクチャの解説を行っていく。今回はメモリ保護や仮想記憶を実現するMMUを取り上げてみよう。

●はじめに

この連載（になるか？）は，コンピュータのアーキテクチャについて知ったかぶりをするために，アーキテクチャに関して直感的に理解することを目標としている。その第1回目は仮想記憶およびMMUについて取り上げる。

MMUとはMemory Management Unitの略語である。つまり，メモリ管理ユニットのことで，MPUの外部または内部にあって仮想記憶機能を実現する。単にC言語などでプログラミングするだけなら，仮想記憶の知識なんてほとんど必要ない。しかし，プログラムのサイズが，ひと昔前に比べて，馬鹿でかくなっており，また，マルチタスクが当然のように行われている昨今，その裏方にはMMUという働き者がいることを心に留めてほしい。

●仮想記憶とは

その昔，まだメモリが高価だった頃，コンピュータに実装できるメモリ容量はわずかなものだった。ときとしてアプリケーションプログラムの容量は実際の物理メモリの容量を超え，そのようなプログラムを動作させるためにはアプリケーションプログラムの側で細工をする必要があった。

プログラムの性質として見ると，ある瞬間瞬間に実行されているのは全体の一部分にすぎない。そこで，プログラムをいくつかのブロックに分割し，必要な部分だけをメモリにロードして実行させ，不要になったらそのブロックを補助記憶装置（多くの場合，ハードディスク）に退避し，代わりにほかの必要なブロックを補助記憶装置から取り出して，新しいブロックと入れ替える仕組みが必要になる（図1）。しかし，実装されている物理メモリの容量を考慮しながらプログラミングをするのは効率的でないし，物理メモリの容量が変化すると同じプログラムが使用できなくなってしまう。そこで，そのようなメモリ管理をOSに任せる仕組みが考案された。これが仮想記憶の原点である。仮想記憶を利用すると，ユーザーは物理メモリを意識することなく，物理メモリの容量を超えるような巨大なプログラムを実行できる。

現在のコンピュータはそのほとんどがマルチタスクで動作している。マルチタスクとは複数のタスク（プログラム）を同時に物理メモリに置き，ある決められた順番に少しずつ（その多くは時分割で）実行していくものである。この場合，各タスクが必要とする全部の領域を物理メモリに割り当てようとすると，メモリに入り切らなくなってしまう。物理的に限られた容量しかないメモリを，多くのタスク間で分割して使用する手段が必要である。この場合も仮想記憶が有効である。というか，仮想記憶といえば，現在ではマルチタスクを実現する手法として紹介されることが多い。

マルチタスクも，物理メモリを複数のブロックに分けて，そのブロックを各タスクに割り当てて実行させることで実現される。このような仮想記憶を行う場合，各タスクが自身に割り当てられた物理メモリのブロック以外をアクセスしないように保護する機能も必要になってくる。

仮想記憶の方式としては，大きく分けて，セグメント方式とページング方式がある。現在ではページング方式が主流なので，ここではページング方式を念頭に置いて話を進める。この場合，タスクのアドレス空間を分割したブロックをページと呼ぶ。また，不要になったページを補助記憶装置に退避し，必要なページを補助記憶装置から復元する作業をページスワップと呼ぶ。

メモリのアクセス速度に比べてハードディスクのアクセス速度は非常に遅いので，ページスワップが頻繁に発生すると，プログラムの実行速度はかなり低下する。しかし，プログラムとデータには，ある程度局所性があるため，ページスワップがあまり発生しないことを期待して仮想記憶が実現されている。ところが，頻繁にページの範囲を超えて分岐が発生するプログラムや不連続な大量のデータを参照するプログラムでは，ページスワップの発生する確率が高くなる。このような場合は，そのタスクのページサイズを大きくすることで，ある程度ページスワップを回避できる。このため，MPUによってはタスクごとにページサイズを可変にできるようになっている。

●アドレス変換

プログラミングにおいて「このプログラムは物理アドレスの何番地に割り当てられるから」などと考えてプログラムを作る人は（OS屋などを除き），まずいない。誰もが，自分の書いたプログラムは，たとえば，「0番地から配置され無限の容量を持っている」と考える。つまり，プログラムはそれぞれ固有のアドレス空間を持っている。マルチタスクを行うということは，重複するアドレス空間を持つ複数のプログラム（タスク）を同時に物理メモリに割り当てて実行するということである。このような操作を可能にするためには，プログラムの中で想定されているアドレスを，実際の物理メモリに配置するためのアドレスに読み換える仕組みが必要になる。これがアドレス変換である。

プログラムが想定しているアドレスは仮想アドレス（論理アドレスともいう）と呼ばれ，物理メモリに割り当てられるアドレスを物理アドレス（実アドレスともいう）と呼ぶ。アドレス変換とは仮想アドレスを物理アドレスに変換する作業のことである。プログラムの仮想アドレス空間は一定の容量を持つページに分割される。このページ単位に仮想アドレスから物理アドレスの変換が行われる。

ページのサイズ（容量）はOSによってまちまちである。昔は1ページが2Kバイトのサイズが多かったが，現在は4Kバイトのサイズが多いように感じる。4Kバイトは16進数で表現すれば1000バイトである。私見ではあるが，人間にとってなんとなく切りのいい数値なのでOS屋さんに好まれるのであろう。

それはともかく，仮想アドレスと物理アドレスの対応は物理メモリ上に置かれたアドレス変換テ

ーブルによる。この変換テーブルはページテーブルと呼ばれ，通常4バイトまたは8バイトの大きさのエントリの集まりである。32ビットOSの場合，アドレスは32ビットで表現されるから，最低でも1ページ当たり32ビット（4バイト）の領域が必要である。もっとも，仮想アドレスと物理アドレスは同一ページ内のオフセット（1ページが4Kバイトの場合はアドレスの下位12ビット）は一致するので，必要なビット数はもう少し少なくてよい。しかし，実際には，そのページの保護情報やページスワップのための情報も必要になるし，ワード長（4バイト）またはダブルワード長（8バイト）のほうが（OSの）プログラムで扱いやすいので，ひとつの仮想アドレスに対して4バイトまたは8バイトのページテーブルエントリが用いられる。

さて，仮想アドレスが32ビット，1ページが4Kバイトの場合を考えよう。この場合，仮想アドレスの下位12ビットがページ内オフセット，上位20ビットがページ番号になる。このページ番号をインデックスとしてページテーブルを参照すれば，そのページに対応する物理アドレスを取り出すことができる。

なお，ページテーブルのベースアドレスはタスクごとに固有な値を持っていて，コンテキスト（タスクを性格づける情報）の一部である特権レジスタに格納されている。図2に仮想アドレスから物理アドレスを得る作業の概念図を示す。この図では20ビットのインデックスでページテーブルを参照するので，ページテーブルのエントリ数は1M個必要である。ページテーブルエントリの容量は4バイトまたは8バイトなので，1タスク当たり4Mバイトまたは8Mバイトの物理メモリの容量がページテーブルのために必要になる。

しかし，タスクの持つアドレス空間は32ビット（4Gバイト）のすべての領域を被っているわけではなく，命令，データ，スタックなど，性質の異なる領域ごとにある程度塊を持って存在している。このような場合，1M個のページテーブルエントリをすべて用意するのは不経済である。下手をしたら物理メモリがページテーブルだけであふれてしまうという状況も起こりかねない。そこで，ページテーブルを多段階に分けて参照する方法が考えられている。

この方式では仮想アドレスのページ番号をさらにいくつかの領域に分ける。たとえば，20ビットのページ番号を上位12ビットと下位8ビットに分ける。この場合，上位12ビットをインデックスとして1段目のテーブルを参照し，2段目のテーブル（これがページテーブル）へのベースアドレスを獲得する。そして，下位8ビットをインデックスとして2段目のテーブルを参照し，物理アドレスを獲得する。この概念図を図3に示す。

図2ではページテーブルを直接参照しているので1レベルのページング，図3では2回目でページテーブルを参照しているので2レベルのページングと呼ばれる。アドレス変換においては，2レベルのページングが主流であるが，MC68030や68040では3レベルのページングを行うこともできる。

TLB

MPUが仮想記憶モードで動作している場合，仮想アドレスから物理アドレスへの変換をいちいち物理メモリ上のページテーブルを参照しにいっていたのでは，その処理が命令実行のボトルネックになってしまう。それを避けるためにMPUは内部にTLB（Translation Lookaside Buffer）と呼ばれる変換テーブルを持っている。日本語ではアドレス変換緩衝機構と訳されることが多い。モトローラはATC（Address Translation Cache），

図1　仮想記憶のイメージ

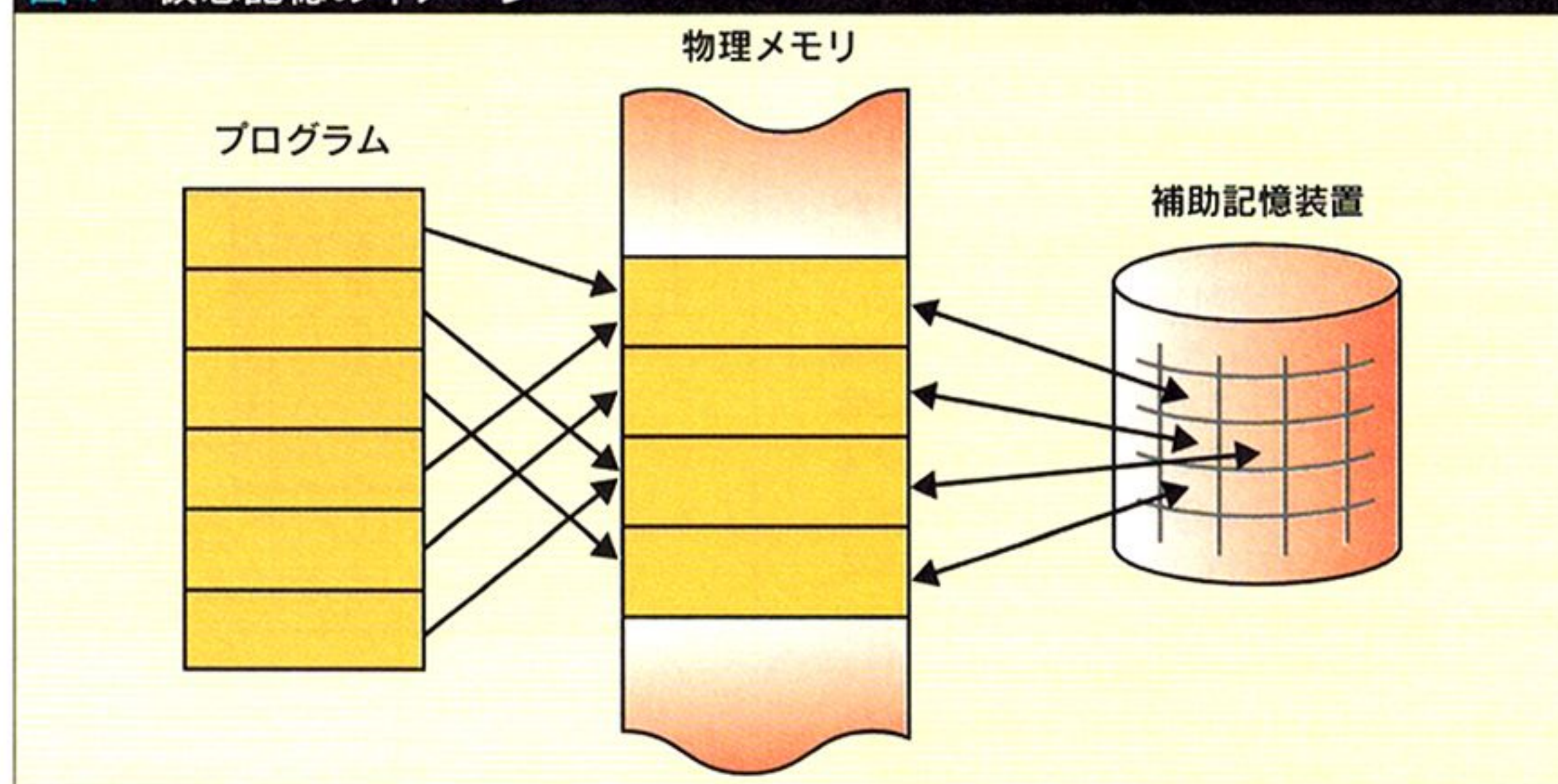



図2　1レベルのアドレス変換

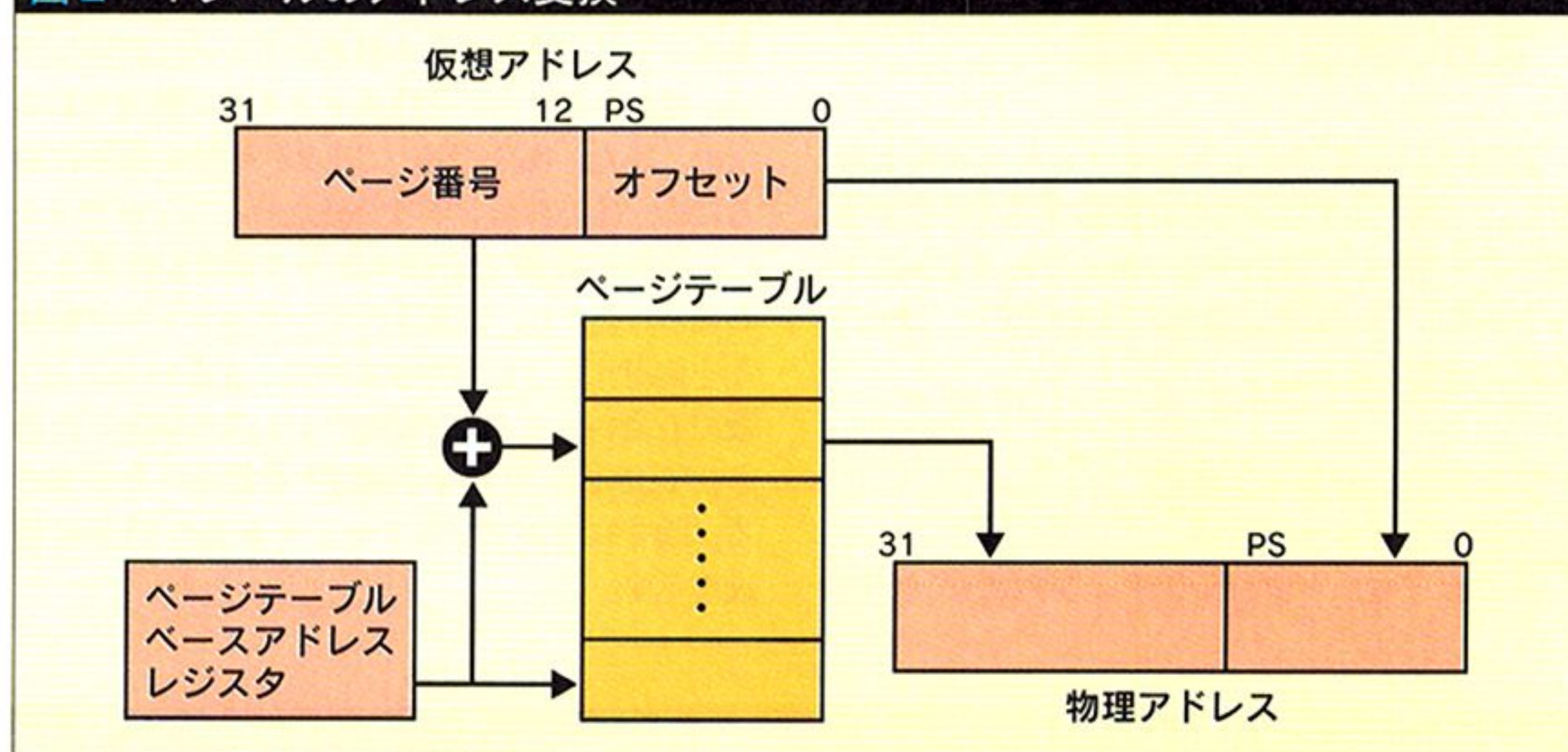
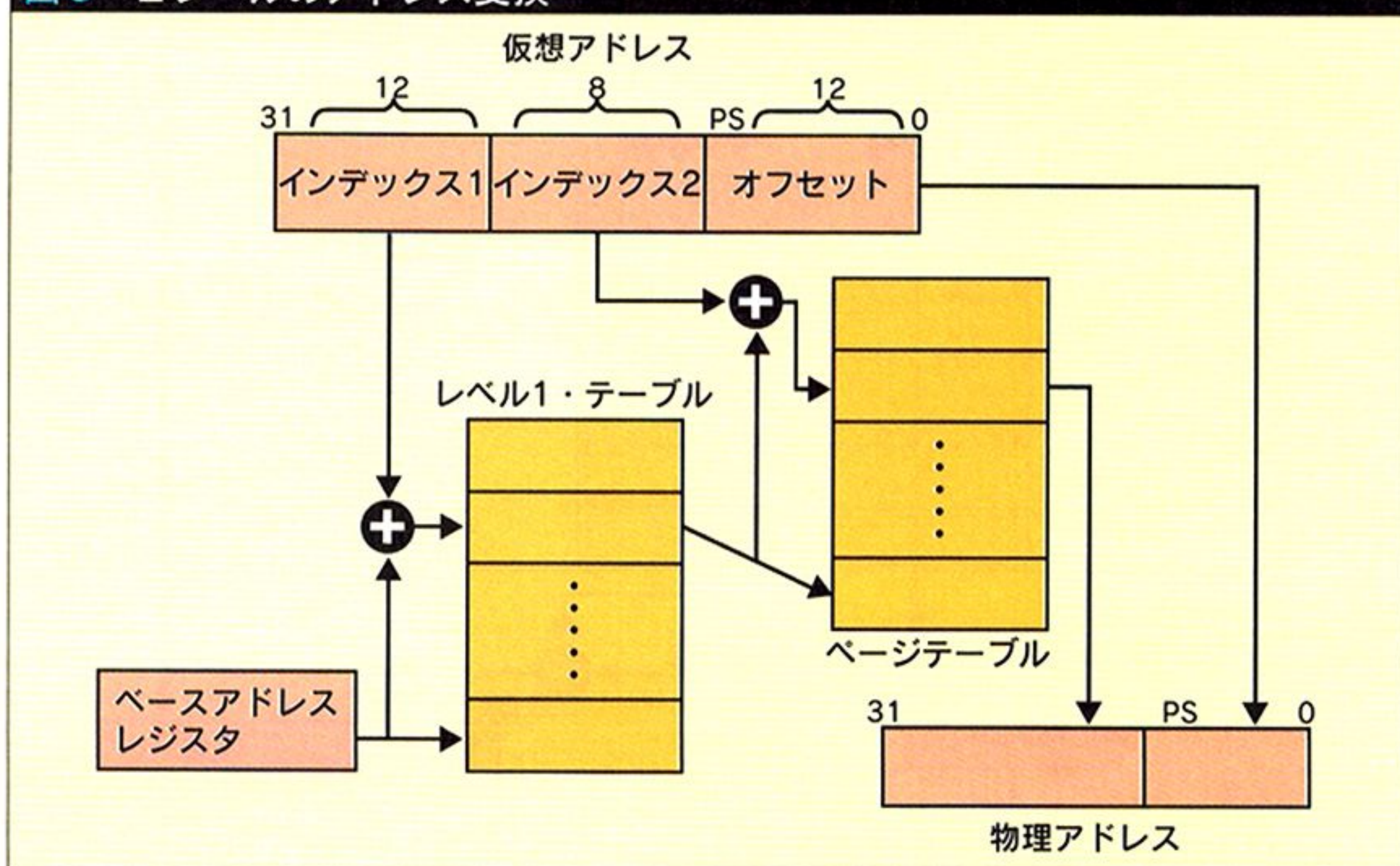



図3　2レベルのアドレス変換

つまり，アドレス変換キャッシュと呼んでいる。その名のとおり，TLBとは，ページテーブルエントリをチップ内にキャッシュしたものである。MPUはアドレス変換を行うとき，まずTLBを参照し，そこに目的の仮想アドレスと物理アドレスのペアが格納されていれば，その物理アドレスを用いて命令を処理する。該当する仮想アドレスがTLB内になければ，物理メモリ上のページテーブルを参照しにいき，その値をTLBに登録する。また，TLBにはページテーブルエントリと同様にメモリ保護などの情報が格納されており，TLB参照の際に不正アクセスがないかチェックする。もし，不正なアクセスである場合はメモリ保護例外を発生する。以上がMMUの機能である。

ただし，最近のRISCチップでは，TLBを参照したとき，仮想アドレスが登録されていないと直ちに例外を発生して，TLBの内容を入れ替える処理をOSのプログラムに任せる。何度もメモリ上のテーブルを参照してTLBの内容を更新する処理は，実現が複雑であり，メモリアクセスはロード/ストア命令だけというRISCのポリシーにも反するし，なによりもパイプライン動作が妨げられてしまう。このため，RISCではTLBの機能そのものがMMUの機能ということもできる。

TLBの構造（連想方式）

TLBとは仮想アドレスをタグとして内容を参照し，一致するタグがあれば対応するデータを物理アドレスとして出力する一種のキャッシュメモリである。その構造は参照の仕方により，次の3種類に分類できる。

- フルアソシアティブ方式
- ダイレクトマップ方式
- Nウェイセットアソシアティブ方式（N≧2）

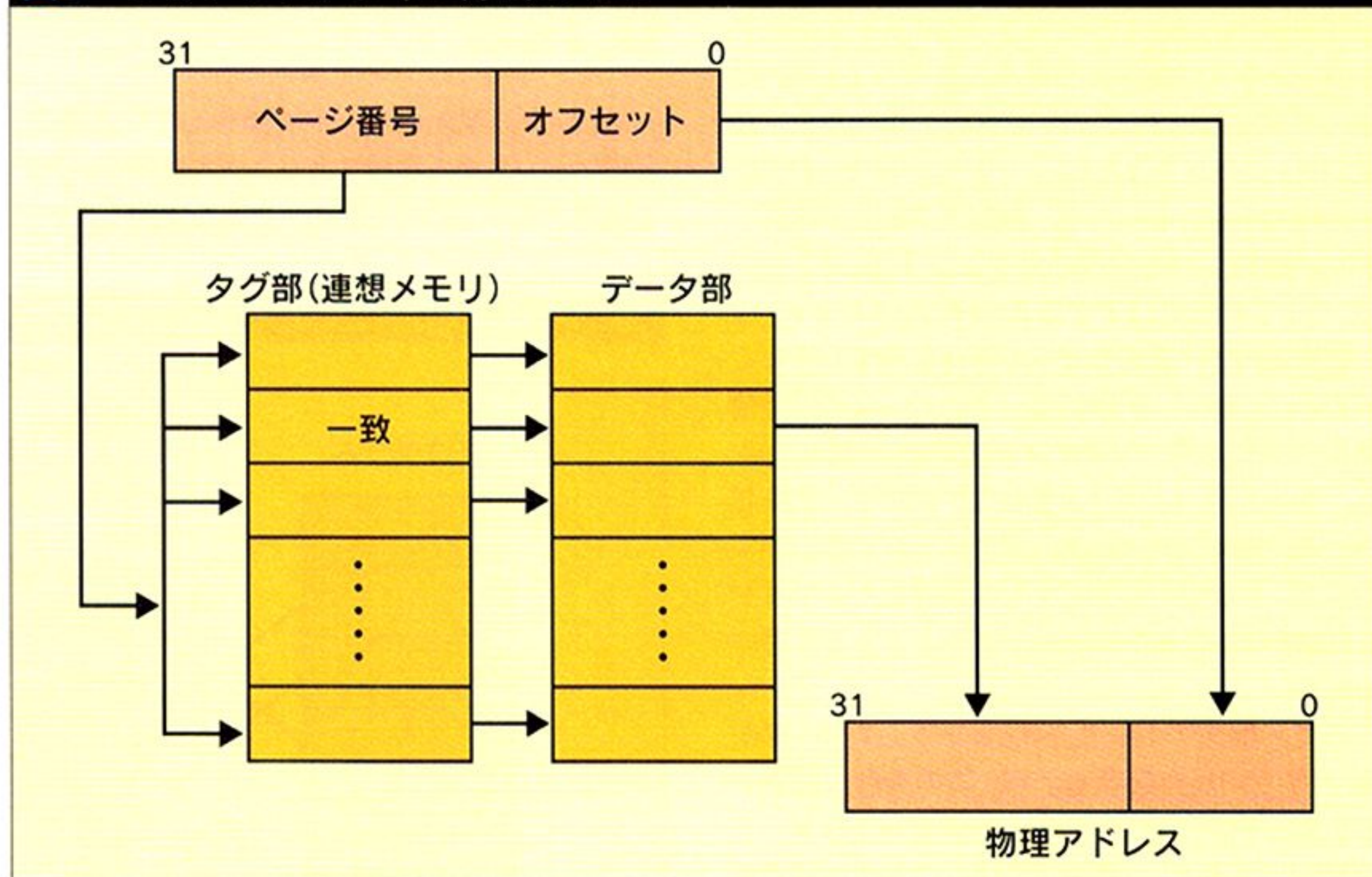
図4　フルアソシアティブ方式のTLB

フルアソシアティブ方式とは，TLBのエントリ数の数だけ異なる仮想アドレスを格納できる方式である。あとで説明するほかの方式とは異なり，各エントリに格納される仮想アドレスに制限はない。連想メモリという特殊なメモリで構成されるため，また，あとで述べるLRU処理が複雑になるため，多くのエントリを持たせることができない。現在の技術では50エントリ程度が限界と思われる。ただし，実装されているエントリを無駄なく使用することができるので，少ないエントリ数でも高いヒット率（仮想アドレスを参照したとき，TLB内に存在する確率）を得ることができる。図4にフルアソシアティブ方式のTLBの構成を示す。

図5　ダイレクトマップ方式のTLB

ダイレクトマップ方式とは，もっとも単純な方式である。仮想アドレスが決まると，その仮想アドレスで参照するエントリが一意に決まってしまう。たとえば，256エントリのダイレクトマップ方式のTLBを参照する方法として，仮想アドレスのビット18～11（8ビット）を使用してエントリをインデックスする方法が考えられる。これは，ページサイズが4Kバイトのときのページ番号の下位8ビットである。8ビットのデータは256種類を識別できるので，仮想アドレスとTLBのエントリを1対1に対応させることができる。ただし，この場合，下位8ビットが一致する仮想アドレスは異なるアドレスであっても同一のエントリが参照されてしまう。プログラムの仮想アドレスが256通りまんべんなく変化することは希なので，場合によっては一度も参照されないエントリが存在する。逆に同じエントリが何度も参照され，前のデータを書き潰してしまう恐れもある。ダイレクトマップ方式は，構造は単純でエントリ数を多く持たせることができるが，エントリ数を多くしないと高いヒット率は期待できない。図5にダイレクトマップ方式のTLBの構成を示す。

Nウェイセットアソシアティブ方式とは，ダイレクトマップ方式の改良版である。ダイレクトマップ方式のエントリをN系統用いて構成する。この方式も，あとで説明するLRU処理の制限からNの値は2または4であることが多い。簡単のために2ウェイセットアソシアティブ方式の場合で説明する。ダイレクトマップの場合と同様に，仮想アドレスが与えられるとエントリは一意に決定されるが，いまの場合は2組のウェイ（エントリの集合）があるので同時に2つのエントリに格納されている仮想アドレスと比較を行う。与えられた仮想アドレスがそのどちらかに一致していればヒットということになる。一般にウェイ数が増えるほどヒット率が向上する。図6に2ウェイセットアソシアティブ方式の

図6　2ウェイセットアソシアティブ方式のTLB

31　仮想アドレス　0
インデックス　オフセット
ウェイ0　タグ部　データ部
ウェイ1　タグ部　データ部
比較器
比較器
セレクタ
31　0
物理アドレス

図7　変換制御レジスタ（TC）

31	28 27	25 24	23	20 19	16
E		SRE	FCL	PS	IS

15	12 11	8 7	4 3	0
TIA	TIB	TIC	TID	

E：変換許可。
SRE：SRP（スーパーバイザルートポインタ）の使用許可
　SRE=0　変換はすべてCRP（CPUルートポインタ）から始まる
　SRE=1　ユーザーアクセスはCRP，スーパーバイザアクセスはSRPを使用
FCL：FC（ファンクションコード）のルックアップ。つまり，最初のデスクリプタへのアクセスをFCの値でインデックスするか。
PS：ページサイズ。1ページのビット数を指定
IS：イニシャルシフト。仮想アドレスのサイズ（32～17ビット）を指定。つまり，アドレス変換時に仮想アドレスの上位をマスクするビット数。32ビットなら0を指定
TIx：テーブルインデックス。TIAが1レベル目，TIBが2レベル目，TICが3レベル目，TIDが4レベル目。それぞれ，仮想アドレスの中で何ビットを占めるかを指定する

注意
IS+TIA+TIB+TIC+TID+PS=32（ビット）の関係を保たなければならない

TLBの構成を示す。

TLBの更新方式

TLBのエントリ数には限りがある。エントリの中に有効なデータが入ってなければ，そこにアドレス変換の情報を格納していけばよいが，新たに変換の情報を登録しようとしたとき，エントリがすでに有効なデータで占められていることがある。この場合は古い情報を追い出して新しい情報を書き込む（上書きする）。

追い出しの対象となるエントリを決定するためにもっとも多く使われるのがLRU（Least Recently Used）という手法である。つまり，時間的にもっとも使用されていないエントリを追い出す。その実現方法は2ウェイセットアソシアティブ方式では簡単である。2つのエントリの組に対して1ビットのLRUビットを設ける。そのビットの値が0か1によって2つのうち対応するエントリをあらかじめ決めておく。そして，0側のウェイがヒットすればLRUを1側に，1側のウェイがヒットすればLRUを0側に更新する。もし，そのエントリに対応する仮想アドレスであって，どちらのエントリの内容とも一致しない仮想アドレスを変換しなければならない場合は，対応する物理アドレスを求め，LRUビットが示す側のウェイのエントリに上書きする。4ウェイセットアソシアティブの場合は4つのエントリに対して6ビットの情報でLRUを構成できるといわれている。フルアソシアティブ方式でのLRUはかなり複雑である。その方式が特許になるくらいややこしいのでここでは説明しない。

現実でも，エントリ数が多いTLBに対してはLRUは用いられない。それでは，フルアソシアティブ方式の場合，追い出すエントリをどのように決定するのか。答えは単純である。適当に決めるのである。具体的には（疑似）乱数を用いてエントリを決定する。これは，どのエントリの仮想アドレスも同じ程度に参照されていると仮定している。どのエントリが選ばれても恨みっこなしということである。

タスク切り替えとTLB

タスクの仮想アドレス空間はタスクごとに固有である。意図的にほかのタスクのアドレス空間の一部を共有させることもあるが，基本的にはコンテキストの一部である，（何回か繰り返し参照して最終的には）ページテーブルのベースアドレスを指し示す，特権レジスタの値によって決定される。このため，タスクが切り替われはTLBの内容も，そのタスクの仮想アドレス空間を反映したものに切り替わらなければならない。論理的にはタスクの数だけTLBが必要ということになる。しかし，現実的には，タスクの数の最大値を予測することは不可能だし，MMU内にいくつものTLBを実装するのは無駄が多い。そこで，多くのMPUではタスクが切り替わるごとにTLBの内容を無効化してしまう。この方式では，タスクが切り替わるたびに同じ仮想アドレスを何回もアドレス変換してしまう可能性があり，それがプログラムの実行速度の低下を招く。

その欠点を回避するために，TLBのタグ部にタスク番号を入れておき，タスク番号込みで仮想アドレスの一致を調べるという方式を採用するMPUもある。この方式だと，タスク番号1の仮想アドレス0番地と，タスク番号2の仮想アドレス0番地が同時にTLBに登録されていても（このような状況が発生するのはフルアソシアティブ方式のTLBに限られるが），2つの0番地を区別することができる。タスクの切り替え時にTLBの内容を無効化する必要もない。

TLBの分離

最近のMPUはパイプライン処理で命令を実行している。命令フェッチやデータアクセスの前には仮想アドレスを物理アドレスに変換する必要があり，そのときTLBが参照される。なにも考えないでMPUを設計すると，ある瞬間では，命令用のアドレス変換でのTLBの参照とデータ用のアドレス変換でのTLBの参照が同時に発生してしまう。命令の仮想アドレスとデータの仮想アドレスは一般には一致しないので，2つの仮想アドレスで同時にTLBを参照することになるが，こ

図8　5レベルのテーブルサーチ

FC　IS　TIA　TIB　TIC　TID　PS
010　・・・・・・・・・　0001　0101　0100　1000　・・・・・・・・・
$2　$1　$5　$4　$8

ルートポインタ
ENTRY #2
ENTRY #1
ENTRY #5
ENTRY #4
ENTRY #8
ページフレームアドレス

FCレベルのテーブル
Aレベルテーブル
（最大8テーブル，16エントリ/テーブル）
Bレベルテーブル
（最大128テーブル，16エントリ/テーブル）
Cレベルテーブル
（最大2Kテーブル，16エントリ/テーブル）
Dレベルテーブル
（最大32Kテーブル，16エントリ/テーブル）

図9　FCルックアップを用いたテーブルサーチ

CPUルートポインタ
（未定義，予約）
4　ユーザーデータ空間
8　ユーザープログラム空間
$C　（ユーザー定義，予約）
$10　（未定義，予約）
$14　スーパーバイザデータ空間
$18　スーパーバイザプログラム空間
$1C　CPU空間（マップなし）

第1テーブルポインタのアドレス＝CPUルートポインタ＋（ファンクションコード×サイズ）

ユーザーデータ空間への分岐
ユーザープログラム空間への分岐
スーパーバイザデータ空間への分岐
スーパーバイザプログラム空間への分岐

このレベルのテーブルインデックスは，変換制御レジスタのTIAフィールドで指定される論理アドレスを使用する

れは不可能である。どちらかの参照を遅れさせて，逐次的に参照することになる。

このときのパイプラインの乱れを嫌って，命令用とデータ用に2つのTLBを採用するMPUもある。キャッシュで命令とデータのデータパスをそれぞれ専用に持たせる構造をハーバードアーキテクチャと呼ぶこともあったが，それのTLB版と考えればよいだろう。実際，ハーバードアーキテクチャを提唱していたのはモトローラだし，モトローラのMPU（68040）などは，命令とデータの2系統のTLBをサポートしている。

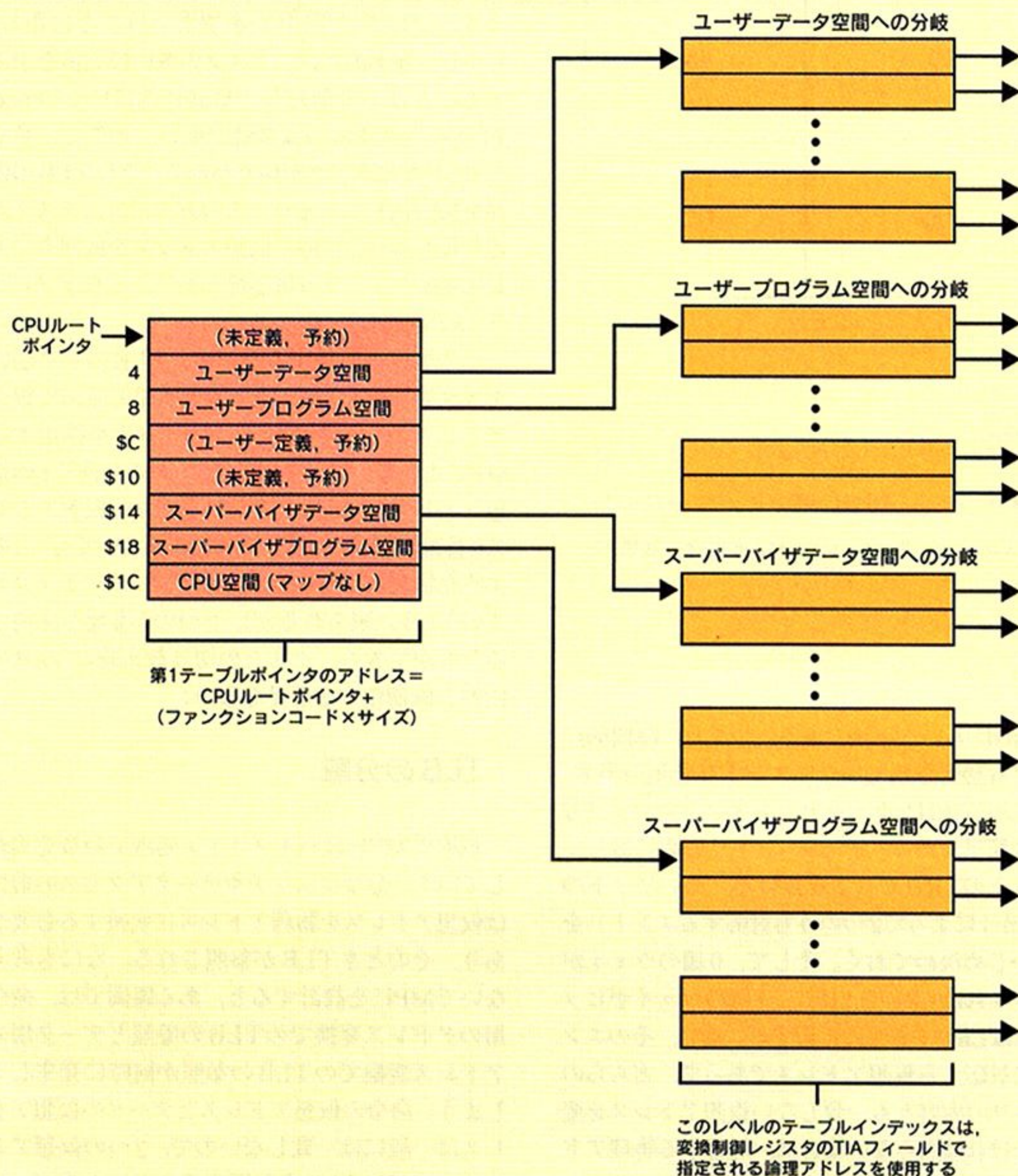
マイクロTLB

命令とデータで同じ規模のTLBを用意するのは大袈裟だし，あまり効果はないように思える。なぜなら，データはともかく，命令のアドレスはシーケンシャルに実行され，分岐で初めて別のアドレスに切り替わる。分岐自身もページサイズの範囲を超えることは希なので，命令のための仮想アドレスを変換しなければならない場合は（データに比べると）極端に少ない。そこで，命令用のTLBとして1～4エントリ程度の特別なTLBを採用するMPUもある。そのようなTLBはマイクロTLBと呼ばれる。多くの場合，マイクロTLBは，本体のTLBの内容をキャッシュしたもので，ページサイズも固定である。命令がマイクロTLBにミスした場合は，まず，本体のTLBを参照し，そこにヒットすれば，そこから物理アドレス情報を持ってきて内容を更新する。TLBのページサイズがマイクロTLBのページサイズよりも大きい場合は，本体のTLBでヒットする確率が高いし，TLBミスに際して物理メモリをアクセスしてTLBの内容を更新するロジックが1系統分で済む。また，マイクロTLBの参照は，本体の巨大なTLBを参照するよりも少ない電力で行えるので，命令だけでなく，データに対しても採用されることもある。

●MMUの実例

ここでは，MMUの実例として，MC68030のMMUを簡単に見てみよう。

アドレス変換

MC68030ではアドレス変換はゼロ～5レベルの範囲で自由に設定できる。その設定を行うためのレジスタが変換制御レジスタ（TC）である。図7にTCの形式を示す。TIA，TIB，TIC，TIDでそれぞれ1レベル，FCルックアップを行えばさらに1レベル増えるので，最大5レベルのページングとなる（インダイレクト指定をすれば6レベルまで可能ということであるが，ここでは触れない）。TIA，TIB，TIC，TIDの値を0に設定することで1～4レベルのページングが可能になる。ゼロレベルというのは，ルートポインタ（CRP，SRP）の中に直接変換後の物理アドレスが指定されている場合（アーリーターミネーションという）

である。MC68030では,

CRP (SRP) →(FC)→TIA→TIB→TIC→TID→

と，変換テーブルをたどっていき（テーブルサーチ），最終的にページテーブルに到達する。5レベルのアドレス変換例を図8に示す。この例ではPSは256バイト(8ビット)で，TIA，TIB，TIC，TIDはそれぞれ4ビット（各テーブルは16エントリ）である。図9にはFCルックアップを用いたテーブルサーチ例を示す。FCによって，次にアクセスするレベルAテーブル（TIAでインデックスされるテーブル）のベースアドレスを個別に設定できるので，メモリ保護が実現できる。

MC68030のMMUでは変換テーブルをサーチするために，仮想アドレスを細かく分割しすぎている感もある。実際的には2～3レベルのページングしか行われないので，オーバースペックともいえる。MC68040ではこの点が改良された。テーブルサーチは3レベルに固定し，ページサイズも8Kバイトまたは4Kバイトのみが許されている。テーブルインデックスはレベルAテーブル，レベルBテーブルが7ビット，レベルCテーブルが5ビット（1ページ8Kバイトの場合），または，6ビット（1ページ4Kバイトの場合）である。MC68030風にいえば,

IS = 0
TIA = 7
TIB = 7
TIC = 5 (8KB/ページ), 6 (4KB/ページ)
PS = 13 (8KB/ページ), 12 (4KB/ページ)

ということになる。

ATC (TLB)

MC68030のATCは22エントリのフルアソシアティブ構成である。仮想アドレスとFCを組で検索し物理アドレスを得る（図10。ただし，この図は機能から推測した予想図である）。一方，MC68040のATCは64エントリの4ウェイセットアソシアティブ構成である。仮想アドレスとFCの最上位ビット（FC[2]）を組で検索し物理アドレスを得る（図11）。FCの最上位しか見ないということは，スーパーバイザとユーザーの区別をするのみで，命令とデータの区別を行わないことを意味する。

●おわりに

MMUというものが，だいたいどのような働きをするものか理解していただいただろうか。以上を覚えておけば知ったかぶりをするには十分であろう。本稿でMMUに興味を持たれた方は専門書で勉強されることをおすすめする。MMUの構成だけを見ても，それを有するMPUの特徴が表われていて面白いのではなかろうか。

図10　MC68030のATC（予想図）

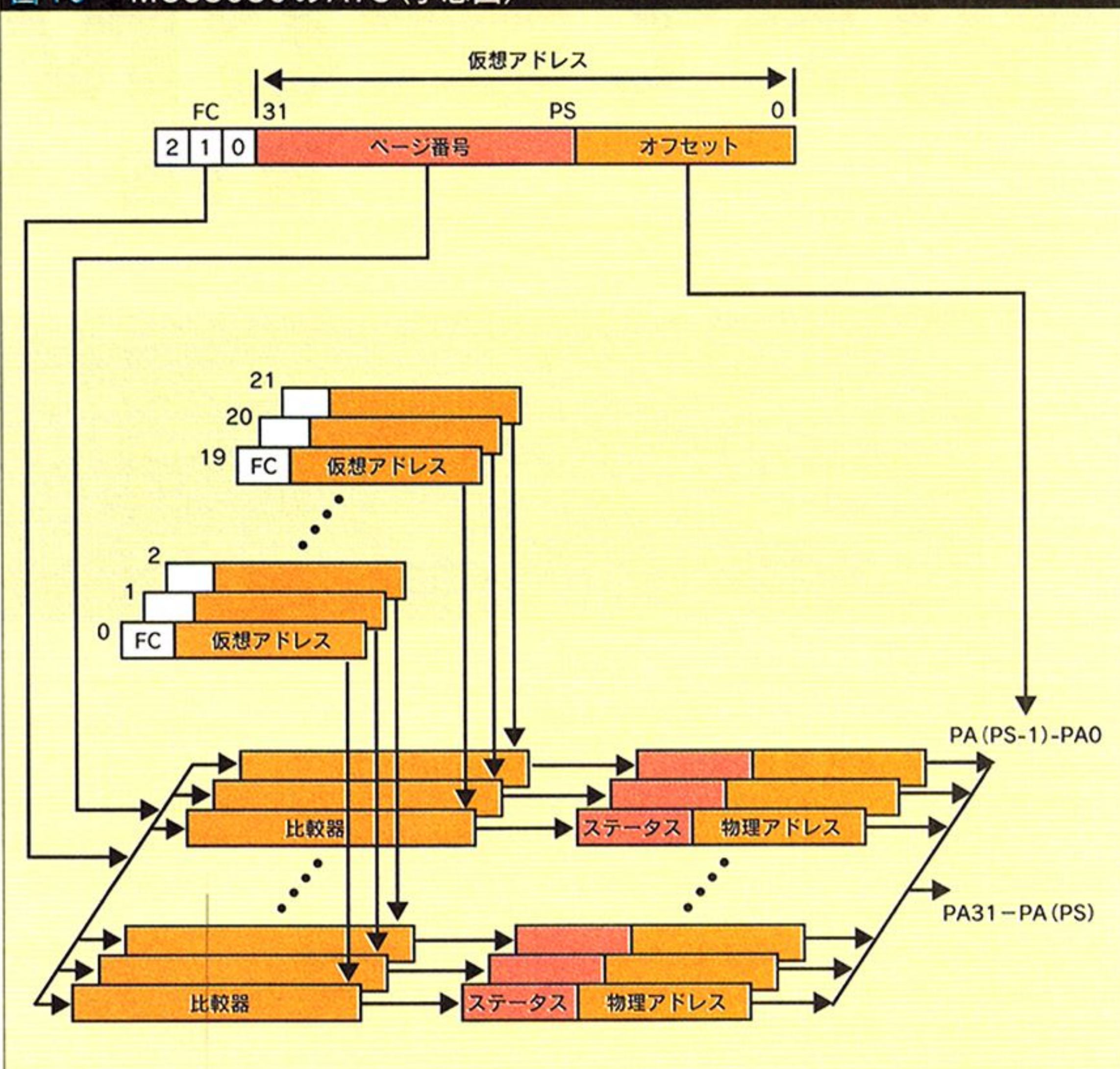
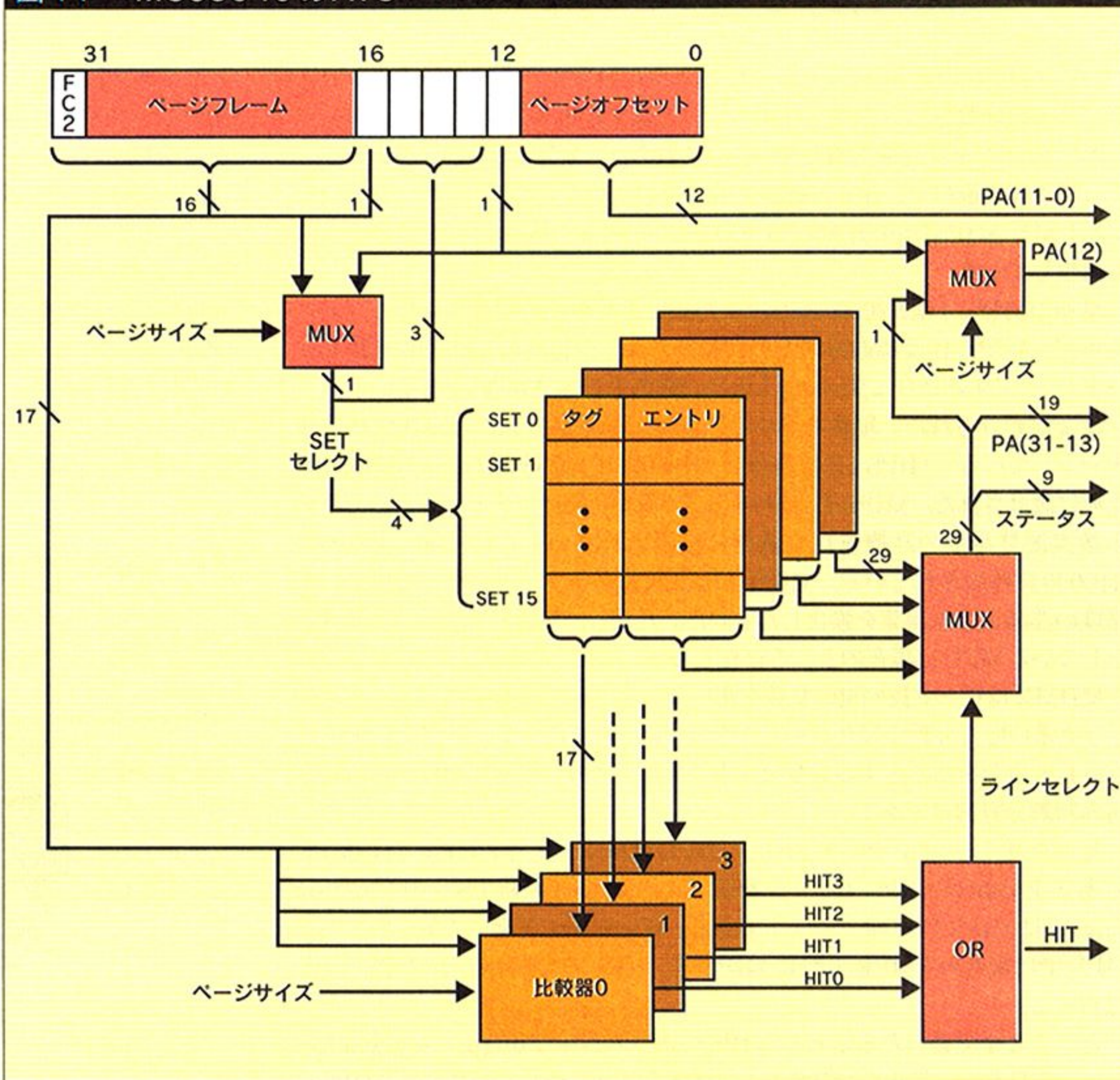



図11　MC68040のATC

初めて読むMIPS

第1回　MIPSの命令セット

中森 章　Nakamori Akira

PlayStationやNintendo64をはじめとして多くのゲームコンソールやWindowsCEマシンで使われるようになってきたMIPSプロセッサ。RISCチップとはいえ，使いこなすにはアセンブラによる記述も必要な局面がある。ここでは，MIPSの命令セットについて解説する。アセンブラ/逆アセンブラをCD-ROMに収録しているのでそちらについても参考にしてほしい。

MIPSといえば，かつてはワークステーションに使用されているMPUという認識が一般的だったが，最近ではPlayStationやNintendo64などのゲーム機やWindowsCEマシンのMPUとして脚光を浴びることが多くなった。ハイエンドから組み込み分野への方針展開というMIPSの戦略が功を奏している。MIPSというアーキテクチャが世間に認知されてきた証拠であろう。

その昔，アセンブリ言語でプログラムを書く人々にとってZ80とMC68000の命令セットを知っておくことが重要だったように，現在ではMIPSの命令セットも教養の一部であるといえるかもしれない。そこで，MIPSの命令に親しんでもらうべく，本稿ではMIPSの命令セットアーキテクチャについて解説してみたい。

MIPS命令セットアーキテクチャ

MIPS RISCの命令セットは，MIPS IからMIPS Vまで，いくつかのカテゴリーに分類される(図1)。IからVへと数字が大きくなるにつれ(ユーザーモードでの)上位互換を有している。これらは，MIPS RISCがバージョンアップされる過程で，徐々に拡張されてきたものだ(図2)。また，その途中でMDMX(MIPSDigital Media Extension)やMIPS16という命令セットも派生している。

最初のMIPS IはR2000/R3000のために設計されたMIPS命令セットの基本だ。MIPS IIはR6000のために設計され，倍精度の浮動小数点数をロード/ストアする命令，Branch Likely(適当な日本語がない)命令，トラップ命令が新設された。現在の多くのR3000の派生品は，このMIPS IIをサポートしている。MIPS IIIは命令セットの64ビット拡張である。R4000のために設計された。MIPS IVはハイエンドRISC向けである。主として浮動小数点演算命令の高機能化を重点に拡張されている。R5000，R8000，R10000に採用されている。MIPS Vは3次元グラフィックなどの用途に単精度の浮動小数点演算を強化したものだ。ただし，MIPS VをインプリメントしているMPUは現在のところ存在しない。

MDMXはビット長の短い整数を用いたベクトル演算を提供する。グラフィック系の応用分野でピクセル単位での計算が可能になる。MIPS16は16ビット長の命令を基本とする命令セットである。メモリ容量に制限がある組み込み制御分野向けである。

身近な例でいえば，PlayStationのMPUはMIPS I，Nintendo64のMPUであるR4300はMIPS III，次世代PlayStation(仮称PS2)のEmotion EngineはMIPS IVのサブセットである。WindowsCE機に搭載されているMPUでは東芝のTMPR3922はMIPS II，NECのVR41xxはMIPS IIIと思われる。

と，ここまで書いたところでMIPSのホームページ(http://www.mips.com/)を見たら，従来のMIPS I～Vは廃止され，新たにMIPS32/MIPS64という命令セットが定義されるらしい。MIPS32はMIPS IIを基本とし，MIPS IVやMIPS Vから有用な命令を取り入れるということだ。MIPS64はMIPS Vを流用したものになるらしい。まあ，命令自体が変更されるわけではないので影響はないけどね。また，本稿では，MIPS Vを実装するMPUがないということで，MIPS IVのレベルで解説を行う。

命令セットアーキテクチャの特徴

MIPSの命令セットはRISC(Reduced Instruction Set Computer)であるから，MC68000系などの高機能な命令やアドレッシングモードを持ったMPUと比較すると，命令の機能は単純化されている。その主な特徴は次のとおり。

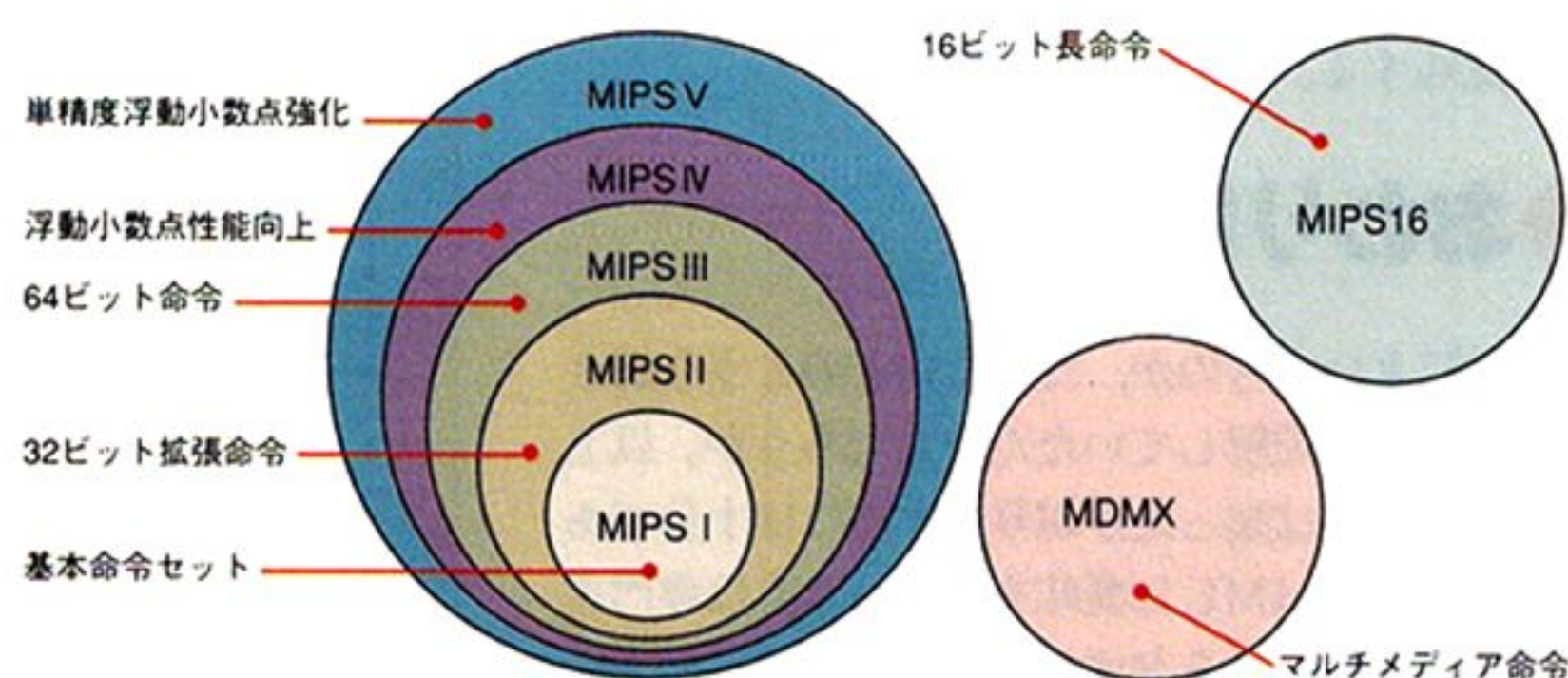
・ロード/ストアアーキテクチャ
・3オペランド命令
・ゼロレジスタと31本の汎用レジスタ
・遅延分岐

以下に詳しく見てみよう。

①ロード/ストアアーキテクチャ

ロード/ストアアーキテクチャはRISCの最大の特徴である。時間のかかるメモリへのアクセスはロード命令，ストア命令のみで行い，それ以外の演算はレジスタ間で行う。メモリへのアドレッシングモードは1種類しかなく，ベースレジスタにオフセット(ディスプレースメント)を加えて実効アドレスを計算するという実に単純なものだ。本誌復刊号のSH4の説明のときの例と同

図1　MIPS命令セットアーキテクチャの関係

単精度浮動小数点強化　MIPS V
浮動小数点性能向上　MIPS IV
64ビット命令　MIPS III
32ビット拡張命令　MIPS II
基本命令セット　MIPS I
16ビット長命令　MIPS16
マルチメディア命令　MDMX

じであるが，メモリの内容に1加算する場合にはMIPSでは次のようになる。

```
la      r2,mem      // memのアドレスをr2にロード
lw      r3,0(r2)    // r2が指すメモリの内容をr3にロード
addiu   r3,r3,1     // r3に1を加え，r3に格納
sw      r3,0(r2)    // r3の値をr2が指すメモリにストア
mem:
    .word   0
```

②3オペランド命令

3オペランド命令はMIPSの命令セットの最大の特徴である。ほとんどすべての演算は3つのレジスタ間で行われる。つまり，ソースレジスタ(rs)とターゲットレジスタ(rt)の内容を演算し，デスティネーションレジスタ(rd)に格納する。多くのMPUでは2オペランド方式が採用されており，ターゲットレジスタとデスティネーションレジスタが同じである。この場合，入力となる値の片方を破壊することになり，プログラムの自由度が制限される。コンパイラでは式の最適化のため，演算の内部表現が3オペランドになっていることが多いが，MIPSではコンパイラとの相性を考慮して3オペランドになっているのかもしれない。MIPSの命令がなぜ3オペランドになったのかは，R2000をモデルとしているHenecyとPatterson著の『コンピュータアーキテクチャ』にも理由が述べられてないのではっきりとはわからないが，

```
addu   r3,r4,r5
```

というのは，r4の値とr5の値を加算(Add Unsigned)してr3に格納するという命令である。Unsigned(符号なし)というのは，入力が無符号整数として扱われるということである。加算の結果は符号があろうとなかろうと同一になるので，add命令との違いは，現象的にはオーバーフロー例外が発生するか否かでしかない。Cコンパイラでは原則的に，加減算でオーバーフローは発生しないことになっているので，add命令よりもaddu命令が用いられる。入力の片方はレジスタでなく，即値(イミディエート値)であることもある。この場合は，ソースレジスタの値とイミディエート値を演算してターゲットレジスタに格納する。

図2　命令セットの変遷

バージョン	発表時期	最初に実装されたプロセッサ
MIPS I	1984	R2000，R3000
MIPS II	1990	R6000
MIPS III	1991	R4xxx
MIPS IV	1994	R5xxx，R7xxx，R8000，R10000
MIPS V	1996	まだない

図3　汎用レジスタの割り当て

レジスタ名	ソフトウェア名	使用目的
r0	zero	常に0
r1	at	アセンブラのテンポラリレジスタ
r2，r3	v0，v1	関数の戻り値
r4-r7	a0-a3	関数への引数
r8-r15	t0-t7	テンポラリレジスタ。関数呼び出しで破壊
r16-r23	s0-s7	関数呼び出しで不変なレジスタ
r24，r25	t8，t9	テンポラリレジスタ。関数呼び出しで破壊
r26，r27	k0，k1	OS用に予約済み
r28	gp	大域変数領域のベースアドレス
r29	sp	スタックポインタ
r30	s8(fp)	s0-s7と同様。必要ならフレームポインタ
r31	ra	関数からの戻り値

図4　パイプラインと遅延分岐

分岐命令	IF	RF	EX	DC	WB		
遅延スロット		IF	RF	EX	DC	WB	
分岐先			IF	RF	EX	DC	WB

クロック

分岐遅延

IF：命令フェッチ　RF：レジスタリード　EX：命令実行
DC：データキャッシュ　WB：ライトバック

図5　CPUの命令形式

●I-Type(イミディエート形式)

31　26	25　21	20　16	15　0
op	rs	rt	immediate

●J-Type(ジャンプ形式)

31　26	25　0
op	target

●R-Type(レジスタ形式)

31　26	25　21	20　16	15　11	10　6	5　0
op	rs	rt	rd	sa	func

op：6ビットの命令コード
rs：5ビットのソースレジスタ
rt：5ビットのターゲット(ソース/デスティネーション)レジスタ。分岐条件
immediate：16ビットのイミディエート値。分岐，アドレスディスプレースメント
target：26ビットの無条件分岐ターゲットアドレス
rd：5ビットのデスティネーションレジスタ
sa：5ビットのシフト量
func：6ビットの機能フィールド

簡易アセンブラについて(CD-ROMに収録)

命令だけ解説しても面白くないので，実際にコードに触れてMIPSのアーキテクチャを体験してもらうため，知人の協力を得て1パスの簡易アセンブラを作成した。アセンブラ記述の注意事項は以下のとおり。

(1)疑似命令

.org　.align　.word

をサポートする。.orgはプログラムの開始番地を指定する。.alignは命令の格納されるアドレスを指定した値で整列する。.wordは定数(32ビット)をメモリに埋め込む。.wordに関してはカンマ(,)で定数を区切って複数指定できる。

(2)ラベル

ラベルの宣言は名前の直後にコロン(:)をつけて行う。ラベルの参照にはコロンはつけない。1～255までの数値にコロンをつけるとローカルラベルとみなされる。ローカルラベルの参照は数値の後ろに'b'(backwardの意)，または'f'(forwardの意)をつける。たとえば，10bとは前方にあるもっとも近い10というラベルの参照を意味する。逆に，10fは後方にあるもっとも近い10というラベルの参照を意味する。アセンブラの都合で，同一のローカルラベルの後方参照は16個の命令までしか許していない。

(3)マクロ命令

la　li　move

をサポートする。意味は先に説明してあるので省略する。

(4)ニーモニック

ニーモニックは小文字で記述すること。

(5)コメント

#で始まる行はコメント行である。

(6)使用法

このアセンブラはコマンドライン上で使用し，

asmips -o 出力ファイル　入力ファイル

で使用する。入力ファイルとはアセンブラ記述したファイルである。例を図17に示す。プログラムは「エラトステネスのふるい」である。出力ファイルはアセンブル結果を格納するオブジェクトファイルである。「べた」なバイナリファイルで，ロードされるアドレス情報は含んでいない。デフォルトはリトルエンディアンのオブジェクトを出力する。ビッグエンディアンのオブジェクトがほしいときは，

-big

オプションを指定する。また，

-o 出力ファイル

の指定を省略すると，標準出力にアドレスと命令コードの組を出力する。たとえば，リスト1のプログラムではリスト2のような出力がなされる。

リスト1　サンプルプログラム

```
#
#  ERATOSTHENTS'ES SIEVE
#
        .org    0xa0002000
#
#  Initialize
#
        li      r2,0
        li      r3,999
        li      r4,0xa0003000
        li      r5,1
1:00
        sb      r5,0 (r4)
        addiu   r4,r4,1
        bne     r2,r3,1b
        addiu   r2,r2,1
#
#  Perform
#
        li      r2,0
        li      r3,999
        li      r4,0xa0003000
loop:
        addu    r5,r4,r2
        lb      r5,0 (r5)
        beq     r5,r0,skip
        addu    r5,r2,r2
        addiu   r5,r5,3
        addu    r6,r2,r5
loop2:
        addu    r7,r4,r6
        sb      r0,0 (r7)
        subu    r7,r6,r3
        blez    r7,loop2
        addu    r6,r6,r5
skip:
        bne     r2,r3,loop
        addiu   r2,r2,1
#
#  Finish
#
        break   0
```

リスト2　アセンブラの出力(出力ファイルの省略時)

```
a0002000        34020000
a0002004        340303e7
a0002008        3c04a000
a000200c        34843000
a0002010        34050001
a0002014        a0850000
a0002018        24840001
a000201c        1443fffd
a0002020        24420001
a0002024        34020000
a0002028        340303e7
a000202c        3c04a000
a0002030        34843000
a0002034        00822821
a0002038        80a50000
a000203c        10a00008
a0002040        00422821
a0002044        24a50003
a0002048        00453021
a000204c        00863821
a0002050        a0e00000
a0002054        00c33823
a0002058        18e0fffc
a000205c        00c53021
a0002060        1443fff4
a0002064        24420001
a0002068        0000000d
```

図6　ロード/ストア命令(MIPS Ⅰ,Ⅱ)

命令	形式と説明　op / base / rt / offset
Load Byte	LB rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。アドレス指定されたバイトの内容を符号拡張してレジスタrtにロードする
Load Byte Unsigned	LBU rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。アドレス指定されたバイトの内容をゼロ拡張してレジスタrtにロードする
Load Halfword	LH rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。アドレス指定されたハーフワードの内容を符号拡張してレジスタrtにロードする
Load Halfword Unsigned	LHU rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。アドレス指定されたハーフワードの内容をゼロ拡張してレジスタrtにロードする
Load Word	LW rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。アドレス指定されたワードの内容を，64ビットモード時は符号拡張してレジスタrtにロードする
Load Word Left	LWL rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。アドレス指定されたバイトがワードの左端になるようにアドレス指定されたワードをシフトする。シフトした結果とレジスタrtの内容をマージし，64ビットモード時は符号拡張してレジスタrtにロードする
Load Word Right	LWR rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。アドレス指定されたバイトがワードの右端になるようにワードをシフトする。シフトした結果とレジスタrtの内容をマージし，64ビットモード時は符号拡張してレジスタrtにロードする
Store Byte	SB rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。レジスタrtの最下位バイトの内容をアドレス指定されたメモリにストアする
Store Halfword	SH rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。レジスタrtの最下位ハーフワードの内容をアドレス指定されたメモリにストアする
Store Word	SW rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。レジスタrtの最下位ワードの内容をアドレス指定されたメモリにストアする
Store Word Left	SWL rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。ワードの左端がアドレス指定されたバイトの位置になるようにレジスタrtを右にシフトする。シフトした結果をメモリワードの下位部分にストアする
Store Word Right	SWR rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算してアドレスを生成する。ワードの右端がアドレス指定されたバイトの位置になるようにレジスタrtを左にシフトする。シフトした結果をメモリワードの上位部分にストアする

図6　ロード/ストア命令(MIPS Ⅲ)

命令	形式と説明　op / base / rt / offset
Load Doubleword	LD rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。アドレス指定されたダブルワードの内容をレジスタrtにロードする
Load Doubleword Left	LDL rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。アドレス指定されたバイトがダブルワードの最左端になるようにダブルワードをシフトする。シフトした結果とrtの内容をマージしレジスタrtにロードする
Load Doubleword Right	LDR rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。アドレス指定されたバイトがダブルワードの最右端になるようにダブルワードをシフトする。シフトした結果とrtの内容をマージしレジスタrtにロードする
Load Linked	LL rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。アドレス指定されたワードの内容をレジスタrtにロードし，LLビットを1にセットする
Load Linked Doubleword	LLD rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。アドレス指定されたダブルワードの内容をレジスタrtにロードし，LLビットを1にセットする
Load Word Unsigned	LWU rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。アドレス指定されたワードの内容をゼロ拡張してレジスタrtにロードする
Store Conditional	SC rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。LLビットが1の場合，レジスタrtの下位ワードの内容をアドレス指定されたメモリにストアし，レジスタrtを1にセットする。LLビットが0の場合は，ストアは行わず，レジスタrtに0をクリアする
Store Conditional Doubleword	SCD rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。LLビットが1の場合，レジスタrtの内容をアドレス指定されたメモリにストアし，レジスタrtを1にセットする。LLビットが0の場合は，ストアは行わず，レジスタrtに0をクリアする
Store Doubleword	SD rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。レジスタrtの内容を指定されたアドレスにストアする
Store Doubleword Left	SDL rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。ダブルワードの最左端がアドレス指定されたバイト位置になるようにレジスタrtの内容をシフトする。シフトした結果をメモリ中のダブルワードの下位部分にストアする
Store Doubleword Right	SDR rt, offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。ダブルワードの最右端がアドレス指定されたバイト位置になるようにレジスタrtの内容をシフトする。シフトした結果をメモリ中のダブルワードの下位部分にストアする

```
    addiu  r3,r4,0x1234
```

は，r4の値にイミディエート値の0x1234を加算(Add Immediate Unsigned)してr3に格納する。

③ゼロレジスタと31本の汎用レジスタ

一般に，MIPSアーキテクチャは32本の汎用レジスタを持っているといわれる。しかし，これは厳密には正しくない。32本のレジスタのうち，0番(r0)はゼロレジスタと呼ばれ，値が0に固定されている。値が0に固定されているほかは任意のオペランドとして使用できる(書き込みもできるが，あまり意味はない)ので，汎用レジスタとみなしても構わないのだが，(0以外の値の)データの保持ができない点で，汎用と呼ぶには抵抗がある。それはともかく，ゼロレジスタはどのような用途に使われるのだろう。もっとも頻度の高い使用法はゼロとの比較である。MIPS(のCPU)には条件フラグというものはなく，条件分岐は指定されたレジスタの値に従って分岐/不分岐が決定される。条件セット命令(これはMC680X0にもある)で条件の成立をテストし結果(0か1)をレジスタにセットする。それとゼロとの比較を行って分岐判定をするわけだ。具体的には，

```
    li      r5,9           // r5に9を格納

    loop:
      ⋮
    bne     r5,r0,loop  // r5の値が0でないとき分岐
    addiu   r5,r5,-1    // r5の値を1減じる。遅延スロット
```

は，r5の値が9から0までの間，10回のループとなる。また，

```
    loop:
      ⋮
    slt    r5,r3,r4     // r3<r4 なら r5 が1
    bne    r5,r0,loop   // r5の値が0でないとき分岐
    nop                 // 遅延スロット。何もしない
```

は，r3の値がr4の値よりも小さい間，分岐を繰り返す。ゼロレジスタは0という定数値を生成するためにも用いられる。

```
    or      r3,r4,r0
    addiu   r3,r0,0x123
```

上の例は，r4の値とr0の値の論理和をr3に格納する。いわば，r4からr3への転送命令である。下の例はr0の値とイミディエート値0x123を加算してr3に格納する。これはr3へのイミディエート値のロード命令とみなせる。

汎用レジスタのなかで特殊な役割を持っているのがr31である。これは，関数の呼び出し命令(JAL)によって，戻りアドレスが自動的に格納される。関数からはr31のアドレスにジャンプすることで戻ってくる。

r0とr31以外は本当に汎用なレジスタである。アセンブリ言語でのプログラミングではこれらを自由に使用できる。ただし，Cコンパイラなどではそれぞれのレジスタに役割分担がある(図3)。コンパイラによるコードとリンクして動作するプログラムを作成するときには，これらを考慮する必要がある。

図7-1　イミディエート命令(MIPS Ⅰ，Ⅱ)

命令	形式と説明　op　rs　rt　immediate
Add Immediate	ADDI rt，rs，immediate 16ビットのイミディエートを符号拡張しレジスタrsと加算する。32ビットの結果をレジスタrtに(64ビットモード時符号拡張して)格納する。2の補数オーバーフローが発生すると例外を発生する
Add Immediate Unsigned	ADDIU rt，rs，immediate 16ビットのイミディエートを符号拡張しレジスタrsと加算する。32ビットの結果をレジスタrtに(64ビットモード時符号拡張して)格納する。オーバーフローが発生しても例外を発生しない
Set On Less Than Immediate	SLTI rt，rs，immediate 16ビットのイミディエートを符号拡張し符号付き整数としてレジスタrsと比較する。rsがイミディエートより小さい場合は1を，そうでない場合は0をレジスタrtに格納する
Set On Less Than Immediate Unsigned	SLTIU rt，rs，immediate 16ビットのイミディエートを符号拡張し符号なし整数としてレジスタrsと比較する。rsがイミディエートより小さい場合は1を，そうでない場合は0をレジスタrtに格納する
And Immediate	ANDI rt，rs，immediate 16ビットイミディエートをゼロ拡張してレジスタrsとANDをとり，結果をレジスタrtに格納する
Or Immediate	ORI rt，rs，immediate 16ビットイミディエートをゼロ拡張してレジスタrsとORをとり，結果をレジスタrtに格納する
Exclusive Or Immediate	XORI rt，rs，immediate 16ビットイミディエートをゼロ拡張してレジスタrsとEx-ORをとり，結果をレジスタrtに格納する
Load Upper Immediate	LUI rt，immediate 16ビットイミディエートを16ビット左にシフトし，ワードの下位16ビットを0にする。結果をレジスタrtに(64ビットモード時符号拡張し)格納する

図7-1　ALUイミディエート命令(MIPS Ⅲ)

命令	形式と説明　op　rs　rt　immediate
Doubleword Add Immediate	DADDI rt，rs，immediate 16ビットのイミディエートを64ビットに符号拡張し，レジスタrsと加算する。64ビットの結果をレジスタrtに格納する。整数オーバーフローが発生すると例外を発生する
Doubleword Add Immediate Unsigned	DADDIU rt，rs，immediate 16ビットのイミディエートを64ビットに符号拡張し，レジスタrsと加算する。64ビットの結果をレジスタrtに格納する。オーバーフローが発生しても例外を発生しない

図7-1　ロード/ストア命令(MIPS Ⅳ)

命令	形式と説明　op　base　hint　offset
Prefetch	PREF　hint,offset(base) 符号拡張したoffsetをレジスタbaseの内容に加算しアドレスを生成する。hintが示す情報に従い，アドレスで指定されたデータをキャッシュにプリフェッチする

図7-2　3オペランド命令(MIPS Ⅰ，Ⅱ)

命令	形式と説明　op　rs　rt　rd　sa　func
Add	ADD rd，rs，rt レジスタrsとrtの内容を加算し，32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する。整数オーバーフローが発生すると例外を発生する
Add Unsigned	ADDU rd，rs，rt レジスタrsとrtの内容を加算し，32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する。整数オーバーフローが発生しても例外を発生しない
Subtract	SUB rd，rs，rt レジスタrsからレジスタrtの内容を減算し，32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する。整数オーバーフローが発生すると例外を発生する
Subtract Unsigned	SUBU rd，rs，rt レジスタrsからレジスタrtの内容を減算し，32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する。整数オーバーフローが発生しても例外を発生しない
Set On Less Than	SLT rd，rs，rt レジスタrsとrtの内容を符号付き整数として比較する。レジスタrsがrtより小さい場合は1を，そうでない場合は0をレジスタrdに格納する
Set On Less Than Unsigned	SLTU rd，rs，rt レジスタrsとrtの内容を符号なし整数として比較する。レジスタrsがrtより小さい場合は1を，そうでない場合は0をレジスタrdに格納する
And	AND rd，rs，rt レジスタrsとrtの内容をビット単位でANDをとり，結果をレジスタrdに格納する
Or	OR rd，rs，rt レジスタrsとrtの内容をビット単位でORをとり，結果をレジスタrdに格納する
Exclusive Or	XOR rd，rs，rt レジスタrsとrtの内容をビット単位でEx-ORをとり，結果をレジスタrdに格納する
Nor	NOR rd，rs，rt レジスタrsとrtの内容をビット単位でNORをとり，結果をレジスタrdに格納する

図7-2　3オペランド命令(MIPS Ⅲ)

命令	形式と説明　op　rs　rt　rd　sa　func
Doubleword Add	DADD rd，rs，rt レジスタrsとrtの内容を加算し，64ビットの結果をレジスタrdに格納する。整数オーバーフローが発生すると例外を発生する
Doubleword Add Unsigned	DADDU rd，rs，rt レジスタrsとrtの内容を加算し，64ビットの結果をレジスタrdに格納する。整数オーバーフローが発生しても例外を発生しない
Doubleword Subtract	DSUB rd，rs，rt レジスタrsからレジスタrtの内容を減算し，64ビットの結果をレジスタrdに格納する。整数オーバーフローが発生すると例外を発生する
Doubleword Subtract Unsigned	DSUBU rd，rs，rt レジスタrsからレジスタrtの内容を減算し，64ビットの結果をレジスタrdに格納する。整数オーバーフローが発生しても例外を発生しない

図7-2　3オペランド命令(MIPS Ⅳ)

命令	形式と説明　op　rs　rt/cc　rd　sa　func
※1	MOVT　rd，rs，cc 浮動小数点演算条件ccが1なら，レジスタrsの内容をレジスタrdに格納する
※2	MOVF　rd，rs，cc 浮動小数点演算条件ccが0なら，レジスタrsの内容をレジスタrdに格納する
※3	MOVZ　rd，rs，rt レジスタrtの値が0なら，レジスタrsの内容をレジスタrdに格納する
※4	MOVN　rd，rs，rt レジスタrtの値が0以外なら，レジスタrsの内容をレジスタrdに格納する

※1　Move Conditional on Floating Point True
※2　Move Conditional on Floating Point False
※3　Move Conditional on Zero
※4　Move Conditional on Not Zero

④遅延分岐

遅延分岐はパイプラインを有効利用しようとするRISCの最大の特徴である(なんか，このフレーズばっか)。通常，分岐先の命令フェッチは実行ステージでの分岐アドレスの計算または条件比較のあとでなければ実行できない。つまり，命令の実行ステージから分岐先の命令フェッチまでに最低1クロックの空きが生じる。これが分岐遅延である。また，この空きを遅延スロットという。従来のCISCではこの部分に実行される命令を意図的に抹殺していた。最終的にパイプラインの実行が分岐遅延の分だけ無駄になる。RISCでは命令を抹殺する制御の代わりに，遅延スロットの命令(分岐命令の直後の命令)を積極的に実行する。その分，パイプラインが有効に実行される。図4はR4300のパイプラインを示している。分岐遅延は1クロックである。

さて，MIPS IIではBranchLikely命令が導入された。無理やり，日本語に訳すと「分岐する傾向がある分岐命令」となる。これは分岐命令が実際に分岐するときのみ遅延スロットの命令を実行するというものだ。分岐しない場合は遅延スロットの命令は無視される。この命令は，遅延スロットに分岐先の先頭の1命令を置くという使い方をする。分岐する傾向があるのだから，このような命令配置にすることで高い確率で分岐遅延をなくすことができる。

CPU命令セットの概要

CPUの命令長はすべて32ビットで，命令形式には図5に示す3種類がある。命令形式を3種と単純化することで，命令のデコードを簡略化し，高い周波数での実行を可能にする。複雑な命令機能やアドレッシングは複数の命令を用いて実行できるようになっている。

実際には，CPUの命令は次の6つのクラスに分類できる。

①ロード/ストア命令
②演算命令

③ジャンプ/分岐命令
④特殊命令
⑤コプロセッサ命令
⑥システム制御コプロセッサ(CP0)命令

このうち，②の演算命令は次の4つに分類される。

①イミディエート命令
②3オペランド命令
③シフト命令
④乗除算命令

これらの命令の概要を図6～図11に示す。また，CPU命令のオペコードマップを図12に示す。

マクロ命令

MIPS RISCのCPU命令は図6～図12に示されるとおりだが，勘の鋭い人ならなにかが足りないことに気づくだろう。これらの図の中にはNOP命令やメモリのアドレスをレジスタにロードする命令は含まれていない。これらの命令はほかの命令や命令の組み合わせで実現可能だからだ。RISCというポリシー上，冗長な命令はサポートされていないのだ。しかし，アセンブリ言語でプログラムを書く場合は不便である。そこで，たいていのアセンブラは，ユーザーが要求する命令で，命令セットに含まれない命令をマクロ命令(疑似命令)としてサポートしている。マクロ命令で主なものを以下に説明する。

①～④はほとんどすべてのアセンブラでサポートされている。⑤，⑥に関してはMIPSの純正アセンブラのみのサポートかもしれないので悪しからず。

① NOP(No Operation)

これはなにもしない疑似命令である。NOPはデスティネーションオペランドをr0にすれば実現できるので，いろいろなバリエーションが考えられる。たとえば，

```
add r0,r0,r0
```

が考えられる。実際には，命令コードが0x00000000になって切りがいい(?)，

```
sll r0,r0,r0
```

が使用される。

② MOVE(Move)

これは，レジスタの値を一方から他方へ移動する疑似命令である。

```
move r3,r4
```

という記述は，通常，

```
or   r3,r0,r4
```

と展開される。MIPS IVでは条件転送命令があるので，同じ動作は，

```
movz r3,r4,r0
```

と記述することもできる。しかし，こちらは疑似命令ではなく，正真正銘CPUの命令である。

③ LI(Load Immediate)

これは，レジスタにイミディエート値をロードする疑似命令である。通常は，

```
lui + ori
```

または，

```
lui + addiu
```

によって実現される。すなわち，32ビットデータを上位16ビットと下位16ビットに分割して，2回に分けてレジスタに格納する。命令コードのイミディエートフィールドが16ビットなのでそうなっている。たとえば，イミディエート値0x12345678をr2にロードする疑似命令である

```
li   r2,0x12345678
```

は，

```
lui  r2,0x1234
ori  r2,r2,0x5678
```

または，

```
lui   r2,0x1234
addiu r2,r2,0x5678
```

と展開される。イミディエート値によってはlui，ori，addiuが単体で用いられる。たとえば，

```
li   r2,0x80000000
```

は，lui命令ではレジスタの下位16ビットに0が格納されることを考えれば，

```
lui  r2,0x8000
```

で十分だし，

```
li   r2,0x1234
```

は，値が16ビット長に収まるので，

```
ori  r2,r0,0x1234
```

または，

```
addiu r2,r0,0x1234
```

で十分である。ここら辺の選択はアセンブラが最適になるようにやってくれるので，liがどのような命令列に展開されるのかは，ユーザーは特に知る必要はない。

簡易逆アセンブラについて(CD-ROMに収録)

アセンブラだけではありがたみがないので，逆アセンブラも作成してみた。使用法は，

```
dismips -saddr=開始アドレス　入力ファイル
```

である。入力ファイルは「ベタ」なバイナリファイルを想定している。

```
-saddr
```

というオプションで開始アドレス(ロードされるアドレス)を16進数で指定する。たとえば，アセンブラのサンプルで挙げたリスト1のプログラム(ファイル名をsieve.sとする)を前述の簡易アセンブラでアセンブルして作られたオブジェクトファイルをsieve.binとすると，その逆アセンブル結果は，

```
asmips -o sieve.bin   sieve.s
dismips  -saddr=a0002000   sieve.bin
```

を実行することで，逆アセンブル結果が標準出力に出力される。その結果をリスト3に示す。-saddrオプションを省略すると，

```
-saddr=0
```

が指定されたものとして逆アセンブルする。デフォルトではリトルエンディアンのオブジェクトを逆アセンブルすることを想定しているが，ビッグエンディアンのオブジェクトを逆アセンブルする場合は，

```
-big
```

オプションを指定する。また，逆アセンブルはMIPS IVの範囲で行われるが，MIPS I～IVで重複する命令コードがある場合は，

```
-mips1
-mips3
-mips4
```

を指定することで命令セットを明示的に指定できる。デフォルトは，

```
-mips3
```

が指定されているものとみなす。COP1X系列の命令を逆アセンブルするためには，

```
-mips4
```

オプションの指定が必要である(デフォルトではコプロセッサ3として逆アセンブルされるため)。

余談であるが，逆アセンブルするオブジェクトファイルの形式は特に想定していないので，PlayStationやNintendo64のプログラムをなんとか吸い出すことができれば，それらのオブジェクトも逆アセンブルできるはずである(多少，ゴミが出力されるかもしれないが)。違法な目的には使用しないように。

リスト3　逆アセンブル結果

```
a0002000    34020000    : ori r2,r0,0x0000
a0002004    340303e7    : ori r3,r0,0x03e7
a0002008    3c04a000    : lui r4,0xa000
a000200c    34843000    : ori r4,r4,0x3000
a0002010    34050001    : ori r5,r0,0x0001
a0002014    a0850000    : sb  r5,0 (r4)
a0002018    24840001    : addiu   r4,r4,0x0001
a000201c    1443fffd    : bne r2,r3,0xa0002014
a0002020    24420001    : addiu   r2,r2,0x0001
a0002024    34020000    : ori r2,r0,0x0000
a0002028    340303e7    : ori r3,r0,0x03e7
a000202c    3c04a000    : lui r4,0xa000
a0002030    34843000    : ori r4,r4,0x3000
a0002034    00822821    : addu    r5,r4,r2
a0002038    80a50000    : lb  r5,0 (r5)
a000203c    10a00008    : beq r5,r0,0xa0002060
a0002040    00422821    : addu    r5,r2,r2
a0002044    24a50003    : addiu   r5,r5,0x0003
a0002048    00453021    : addu    r6,r2,r5
a000204c    00863821    : addu    r7,r4,r6
a0002050    a0e00000    : sb  r0,0 (r7)
a0002054    00c33823    : subu            r7,r6,r3
a0002058    18e0fffc    : blez    r7,r0,0xa000204c
a000205c    00c53021    : addu    r6,r6,r5
a0002060    1443fff4    : bne r2,r3,0xa0002034
a0002064    24420001    : addiu   r2,r2,0x0001
a0002068    0000000d    : break   0
a000206c    ffffffff    : sd  r31,-1 (r31)
```

図7-3　シフト命令(MIPS I，II)

命令	形式と説明　op　rs　rt　rd　sa　func
Shift Left Logical	SLL rd, rt, sa レジスタrtの内容をsaビット左にシフトし，下位ビットに0を挿入する。32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する
Shift Right Logical	SRL rd, rt, sa レジスタrtの内容をsaビット右にシフトし，上位ビットに0を挿入する。32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する
Shift Right Arithmetic	SRA rd, rt, rs レジスタrtの内容をsaビット右にシフトし，上位ビットを符号拡張する。32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する
Shift Left Logical Variable	SLLV rd, rt, rs レジスタrtの内容を左にシフトし，下位ビットに0を挿入する。シフトするビット数はレジスタrsの下位5ビットで指定する。32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する
Shift Right Logical Variable	SRLV rd, rt, rs レジスタrtの内容を右にシフトし，上位ビットに0を挿入する。シフトするビット数はレジスタrsの下位5ビットで指定する。32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する
Shift Right Arithmetic Variable	SRAV rd, rt, rs レジスタrtの内容を右にシフトし，上位ビットを符号拡張する。シフトするビット数はレジスタrsの下位5ビットで指定する。32ビットの結果を(64ビットモード時符号拡張して)レジスタrdに格納する

図7-3　シフト命令(MIPS III)

命令	形式と説明　op　rs　rt　rd　sa　func
Doubleword Shift Left Logical	DSLL rd, rt, sa レジスタrtの内容をsaビット左にシフトし，下位ビットに0を挿入する。64ビットの結果をレジスタrdに格納する
Doubleword Shift Right Logical	DSRL rd, rt, sa レジスタrtの内容をsaビット右にシフトし，上位ビットに0を挿入する。64ビットの結果をレジスタrdに格納する
Doubleword Shift Right Arithmetic	DSRA rd, rt, sa レジスタrtの内容をsaビット右にシフトし，上位ビットを符号拡張する。64ビットの結果をレジスタrdに格納する
Doubleword Shift Left Logical Variable	DSLLV rd, rt, rs レジスタrtの内容を左にシフトし，下位ビットに0を挿入する。シフトするビット数はレジスタrsの下位6ビットで指定する。64ビットの結果をレジスタrdに格納する
Doubleword Shift Right Logical Variable	DSRLV rd, rt, rs レジスタrtの内容を右にシフトし，上位ビットに0を挿入する。シフトするビット数はレジスタrsの下位6ビットで指定する。64ビットの結果をレジスタrdに格納する
Doubleword Shift Right Arithmetic Variable	DSRAV rd, rt, rs レジスタrtの内容を右にシフトし，上位ビットを符号拡張する。シフトするビット数はレジスタrsの下位6ビットで指定する。64ビットの結果をレジスタrdに格納する
Doubleword Shift Left Logical＋32	DSLL32 rd, rt, sa レジスタrtの内容を32＋saビット左にシフトし，下位ビットに0を挿入する。64ビットの結果をレジスタrdに格納する
Doubleword Shift Right Logical＋32	DSRL32 rd, rt, sa レジスタrtの内容を32＋saビット右にシフトし，上位ビットに0を挿入する。64ビットの結果をレジスタrdに格納する
Doubleword Shift Right Arithmetic＋32	DSRA32 rd, rt, sa レジスタrtの内容を32＋saビット右にシフトし，上位ビットを符号拡張する。64ビットの結果をレジスタrdに格納する

図7-4　乗除算命令(MIPS I，II)

命令	形式と説明　op　rs　rt　rd　sa　func
Multiply	MULT rs, rt レジスタrsとrtの内容を32ビット符号付き整数として乗算する。64ビットの結果を，特殊レジスタHIとLOに（64ビットモード時符号拡張して）格納する
Multiply Unsigned	MULTU rs, rt レジスタrsとrtの内容を32ビット符号なし整数として乗算する。64ビットの結果を，特殊レジスタHIとLOに（64ビットモード時符号拡張して）格納する
Divide	DIV rs, rt レジスタrsをレジスタrtの内容で除算する。オペランドは32ビット符号付き整数として扱う。32ビットの商を特殊レジスタLOに，32ビットの剰余を特殊レジスタHIに（64ビットモード時符号拡張して）格納する
Divide Unsigned	DIVU rs, rt レジスタrsをレジスタrtの内容で除算する。オペランドは32ビット符号なし整数として扱う。32ビットの商を特殊レジスタLOに，32ビットの剰余を特殊レジスタHIに（64ビットモード時符号拡張して）格納する
Move From HI	MFHI rd 特殊レジスタHIの内容をレジスタrdに転送する
Move From LO	MFLO rd 特殊レジスタLOの内容をレジスタrdに転送する
Move To HI	MTHI rs レジスタrsの内容を特殊レジスタHIに転送する
Move To LO	MTLO rs レジスタrsの内容を特殊レジスタLOに転送する

図7-4　乗除算命令(MIPS III)

命令	形式と説明　op　rs　rt　rd　sa　func
Doubleword Multiply	DMULT rs, rt レジスタrsとrtの内容を符号付き整数として乗算する。128ビットの結果を特殊レジスタHIとLOに格納する
Doubleword Multiply Unsigned	DMULTU rs, rt レジスタrsとrtの内容を符号なし整数として乗算する。128ビットの結果を特殊レジスタHIとLOに格納する
Doubleword Divide	DDIV rs, rt レジスタrsをレジスタrtの内容で除算する。オペランドは符号付き整数として扱う。64ビットの商を特殊レジスタLOに，64ビットの剰余を特殊レジスタHIに格納する
Doubleword Divide Unsigned	DDIVU rs, rt レジスタrsをレジスタrtの内容で除算する。オペランドは符号なし整数として扱う。64ビットの商を特殊レジスタLOに，64ビットの剰余を特殊レジスタHIに格納する

④ LA(Load Address)

これは，レジスタにメモリアドレスをロードする疑似命令である。実現方法は上のliと同じである。ただ，異なるのは，luiやoriのイミディエート値を決定するのはリンカであるという点である。

```
    la    r3,mem
```

という記述に対して，アセンブラはとりあえず，

```
    lui    r3,0x0000
    ori    r3,r3,0x0000
```

または，

```
    lui    r3,0x0000
    addiu  r3,r3,0x0000
```

というコードを生成しておき，0x0000の部分にリンカが値を挿入してくれることを期待する。

⑤ ULW(Unaligned Load Word)

これは，データタイプの値(この場合，4バイト)に整列されていないアドレスから1ワード(4バイト)の値をレジスタにロードするための疑似命令である。MIPSアーキテクチャではデータタイプに整列されていないアドレスに対してロード/ストアを実行することは禁止されている(アドレスエラー例外が発生する)のだが，過去のプログラムとのしがらみなどから，例外的なアクセスを余儀なくされる場合に使用される疑似命令である。これは，lwr命令とlwl命令で実現される。たとえば，

```
    ulw  r3,1(r2)
```

という記述は，

```
    lwl  r3,4(r2)   // リトルエンディアンの場合
    lwr  r3,1(r2)
```

または，

```
    lwl  r3,1(r2)   // ビッグエンディアンの場合
    lwr  r3,4(r2)
```

と展開される。

⑥ ROR/ROL(Rotate Right/Rotate Left)

これは，いわゆるローテート命令である。あまり実用性がないので疑似命令となっている。これらは，srlとsllによって実現される。たとえば，

```
    ror   r3,r2,5
    rol   r3,r3,5
```

は，それぞれ，

図8　ジャンプ命令(MIPS I，II)

命令	形式と説明 op / target
Jump	J target 26ビットのターゲットアドレスを左に2ビット分シフトし，PCの上位4ビットと結合したアドレスへ，1命令遅れてジャンプする
Jump And Link	JAL target 26ビットのターゲットアドレスを左に2ビット分シフトし，PCの上位4ビットと結合したアドレスへ，1命令遅れてジャンプする。遅延スロットに続く命令のアドレスをr31(リンクレジスタ)に格納する

命令	形式と説明 op / rs / rt / rd / sa / funct
Jump Register	JR rs レジスタrsのアドレスへ1命令遅れてジャンプする
Jump And Link Register	JALR rs,rd レジスタrsのアドレスへ1命令遅れてジャンプする。遅延スロットに続く命令のアドレスをレジスタrdに格納する

図8　分岐命令(MIPS I，II)

命令	形式と説明 op / rs / rt / offset
Branch On Equal	BEQ rs,rt,offset レジスタrsとrtが等しかった場合，分岐アドレスへ分岐する
Branch On Not Equal	BNE rs,rt,offset レジスタrsとrtが等しくない場合，分岐アドレスへ分岐する
Branch On Less Than Or Equal To Zero	BLEZ rs,offset レジスタrsが0以下の場合，分岐アドレスへ分岐する
Branch On Greater Than Zero	BGTZ rs,offset レジスタrsが0より大きい場合，分岐アドレスへ分岐する

命令	形式と説明 REGIMM / rs / sub / offset
Branch On Less Than Zero	BLTZ rs,offset レジスタrsが0より小さい場合，分岐アドレスへ分岐する
Branch On Greater Than Or Equal To Zero	BGEZ rs,offset レジスタrsが0以上の場合，分岐アドレスへ分岐する
Branch On Less Than Zero And Link	BLTZAL rs,offset 遅延スロットに続く命令のアドレスをレジスタr31(リンクレジスタ)へ格納し，レジスタrsが0より小さい場合，分岐アドレスへ分岐する
Branch On Greater Than Or Equal To Zero And Link	BGEZAL rs,offset 遅延スロットに続く命令のアドレスをレジスタr31(リンクレジスタ)へ格納し，レジスタrsが0以上の場合，分岐アドレスへ分岐する

```
srl   r1,r2,5
sll   r3,r2,(32-5)
```
および,
```
sll   r1,r2,5
srl   r3,r2,(32-5)
```
と展開される。ここでは，アセンブラ用のテンポラリレジスタであるr1が暗黙的に使用されている。普段でもr1はあまり使用しないほうがいいということか。

FPU命令セットの概要

FPUはMIPSのMPUの中ではコプロセッサ1(CP1)として実装されている。R2000，R3000ではR2010，R3010という別チップで供給されていたが，R4000以降は1チップに内蔵された。組み込み制御用にFPUをサポートしないMPUも存在する。それらに外付けのFPUは存在しない(多分)。組み込み用途では浮動小数点演算は不要という神話が根強く生き残っているためと思われる。実際，ほとんどのことは固定小数点数(整数)で代替できてしまうので，本当のことかもしれないが。でも，3次元グラフィックは浮動小数点数でないと不便だろう(復刊1号で丹氏が固定小数点数でもできると書いていた気もするなあ)。

ともかく，MIPSのFPU命令セットについて説明する。CPUと同様に，FPUにも3種類の基本形式がある(図13)。なお，浮動小数点数のデータ形式は有名なIEEE-754の2進表現に準拠している。FPU命令のオペコードマップを図14に示す。

・I-Type(イミディエート形式)はロード/ストア命令などに使用される

図8　分岐命令(MIPS I，II)

命令	形式と説明 op / rs / rt / offset
Branch On Equal Likely	BEQL rs, rt, offset レジスタrsとrtが等しい場合，分岐アドレスへ分岐する。分岐条件が成立しなかった場合，分岐遅延スロット内の命令は破棄される
Branch On Not Equal Likely	BNEL rs, rt, offset レジスタrsとrtが等しくない場合，分岐アドレスへ分岐する。分岐条件が成立しなかった場合，分岐遅延スロット内の命令は破棄される
Branch On Less Than Or Equal To Zero Likely	BLEZL rs, offset レジスタrsが0以下の場合，分岐アドレスへ分岐する。分岐条件が成立しなかった場合，分岐遅延スロット内の命令は破棄される
Branch On Greater Than Zero Likely	BGTZL rs, offset レジスタrsが0より大きい場合，分岐アドレスへ分岐する。分岐条件が成立しなかった場合，分岐遅延スロット内の命令は破棄される

命令	形式と説明 REGIMM / rs / sub / offset
Branch On Less Than Zero Likely	BLTZL rs, offset レジスタrsが0より小さい場合，分岐アドレスへ分岐する。分岐条件が成立しなかった場合，分岐遅延スロット内の命令は破棄される
Branch On Greater Than Or Equal To Zero Likely	BGEZL rs, offset レジスタrsが0以上の場合，分岐アドレスへ分岐する。分岐条件が成立しなかった場合，分岐遅延スロット内の命令は破棄される
Branch On Less Than Zero And Link Likely	BLTZALL rs, offset 遅延スロットに続く命令のアドレスをレジスタr31(リンクレジスタ)に格納する。レジスタrsが0より小さい場合，分岐アドレスへ分岐する。分岐条件が成立しなかった場合，分岐遅延スロット内の命令は破棄される
Branch On Greater Than Or Equal To Zero And Link Likely	BGEZALL rs, offset 遅延スロットに続く命令のアドレスをレジスタr31(リンクレジスタ)に格納する。レジスタrsが0以上の場合，分岐アドレスへ分岐する。分岐条件が成立しなかった場合，分岐遅延スロット内の命令は破棄される

図8　ジャンプ命令(MIPS16)

命令	形式と説明 op / target
Jump And Link Exchange	JALX　target 26ビットのターゲットアドレスを左に2ビット分シフトし，PCの上位4ビットと結合したアドレスへ，1命令遅れてジャンプする。遅延スロットに続く命令のアドレスをr31(リンクレジスタ)に格納する。ジャンプ先の命令はMIPS16として実行する

・R-Type(レジスタ形式)は2または3オペランドの演算命令に使用される
・その他は分岐命令や転送命令に使用される
　M-Typeと記述している文献もある

①ロード/ストア命令

ロード/ストア命令はメモリ上にある浮動小数点数データをFPUレジスタにロードしたり，FPUレジスタのデータをメモリにストアするのに用いられる。通常はベースとなるCPUレジスタにオフセット(ディスプレースメント)を加算してメモリのアドレスを指定するが，MIPS IVではオフセットの代わりにCPUレジスタ(インデックスとして利用)も指定できるようになった。

```
lwc1   fp2,0(r10)
swc1   fp2,0(r10)
```
は32ビット長のデータのロード/ストア，
```
ldc1   fp2,0(r10)
sdc1   fp2,0(r10)
```
は64ビット長のデータのロード/ストアである。lwc1/swc1については，R2000時代からのしがらみで，64ビットの浮動小数点数データを上下32ビットずつに分離してロード/ストアすることも可能であるが，話がややこしくなるのでここでは説明しない。また，
```
lwxc1  fp2,r4(r10)
swxc1  fp2,r4(r10)
ldxc1  fp2,r4(r10)
sdxc1  fp2,r4(r10)
```
はr4をインデックスとするロード/ストア命令の例である。

②演算命令

加減乗除，平方根，逆数，積和演算がある。基本的に3オペランド演算で，転送，平方根，変換，比較は2オペランド，積和演算は4オペランドである。図15にMIPS IVでサポートされている演算機能の一覧を示す。

図9　特殊命令(MIPS Ⅰ，Ⅱ)(1/2)

命令	形式と説明　SPECIAL｜rs｜rt｜rd｜sa｜func
Synchronize	SYNC 現在パイプライン内にあるロード／ストア命令を、新たなロード／ストア命令の実行が開始される前に完結させる
System Call	SYSCALL システムコール例外を発生し、例外処理プログラムに制御を移する
Breakpoint	BREAK ブレークポイント例外を発生し、例外処理プログラムに制御を移する

図9　特殊命令(MIPS Ⅲ)(1/2)

命令	形式と説明　SPECIAL｜rs｜rt｜rd｜sa｜func
Trap If Greater Than Or Equal	TGE rs, rt レジスタrsとrtを符号付き整数として比較し、レジスタrsがrt以上の場合、例外を発生する
Trap If Greater Than Or Equal Unsigned	TGEU rs, rt レジスタrsとrtを符号なし整数として比較し、レジスタrsがrt以上の場合、例外を発生する
Trap If Less Than	TLT rs, rt レジスタrsとrtを符号付き整数として比較し、レジスタrsがrtより小さい場合、例外を発生する
Trap If Less Than Unsigned	TLTU rs, rt レジスタrsとrtを符号なし整数として比較し、レジスタrsがrtより小さい場合、例外を発生する
Trap If Equal	TEQ rs, rt レジスタrsとrtが等しいとき、例外を発生する
Trap If Not Equal	TNE rs, rt レジスタrsとrtが等しくないとき、例外を発生する

図9　特殊命令(MIPS Ⅲ)(2/2)

命令	形式と説明　REGIMM｜rs｜sub｜immediate
Trap If Greater Than Or Equal Immediate	TGEI rs, Immediate レジスタrsの内容と、16ビット符号拡張したイミディエートを符号付き整数として比較し、rsの内容がイミディエート以上のとき、例外を発生する
Trap If Greater Than Or Equal Immediate Unsigned	TGEIU rs, Immediate レジスタrsの内容と、16ビット0拡張したイミディエートを符号なし整数として比較し、rsの内容がイミディエート以上のとき、例外を発生する
Trap If Less Than Immediate	TLTI rs, Immediate レジスタrsの内容と、16ビット符号拡張したイミディエートを符号付き整数として比較し、rsの内容がイミディエートより小さいとき、例外を発生する
Trap If Less Than Immediate Unsigned	TLTIU rs, Immediate レジスタrsの内容と、16ビット0拡張したイミディエートを符号なし整数として比較し、rsの内容がイミディエートより小さいとき、例外を発生する
Trap If Equal Immediate	TEQI rs, Immediate レジスタrsの内容がイミディエートと等しいとき、例外を発生する
Trap If Not Equal Immediate	TNEI rs, Immediate レジスタrsの内容がイミディエートと等しくないとき、例外を発生する

③比較と条件分岐，および条件転送

FPUにも条件フラグはない。その代わりに条件ビット(cc)というビットがある。MIPS IIIまでは，この条件ビットは1ビットであったが，MIPS IVでは8ビットに増加された。FPUの条件分岐命令は，この条件ビットが1であるか0であるかを判定して分岐する。たとえば，

```
c.eq.s  fp2,fp4  // fp2とfp4が等しいか比較。条件ビットはcc0"
bc1t    target   // 条件ビット(cc0)が真(1)ならtargetへ分岐
```

という命令シーケンスになる。条件ビットとしてcc0以外を使用する場合は，

```
c.eq.s  cc3,fp2,fp4
bc1t    cc3,target
```

などと記述する。浮動小数点数の比較の種類を図16に示す。図16でUnorderedとは，片方，または両方のオペランドが非数や無限大だったりして，「比較不能」という関係を表す。

条件転送は，パイプラインの流れを乱す，分岐命令を削減するために

図10　コプロセッサ命令(1/2)(MIPS Ⅰ，Ⅱ)

命令	形式と説明　op｜base｜rt｜offset
Load Word To Coprocessor z	LWCz rt, offset (base) offsetを符号拡張してレジスタbaseに加算し、アドレスを生成する。アドレス指定されたワードの内容を、コプロセッサzの汎用レジスタrtにロードする
Store Word From Coprocessor z	SWCz rt, offset(base) offsetを符号拡張してレジスタbaseに加算し、アドレスを生成する。コプロセッサzの汎用レジスタrtの内容を、アドレス指定されたメモリ位置にストアする

図10　コプロセッサ命令(2/2)(MIPS Ⅰ，Ⅱ)

命令	形式と説明　COPz｜sub｜rt｜rd｜0
Move To Coprocessor z	MTCz rt, rd CPUレジスタrtの内容をコプロセッサzの汎用レジスタrdに転送する
Move From Coprocessor z	MFCz rt, rd コプロセッサzの汎用レジスタrdの内容をCPUレジスタrtに転送する
Move Control To Coprocessor z	CTCz rt, rd CPUレジスタrtの内容をコプロセッサzのコプロセッサ制御レジスタrdに転送する
Move Control From Coprocessor z	CFCz rt, rd コプロセッサzのコプロセッサ制御レジスタrdの内容をCPUレジスタrtに転送する

命令	形式と説明　COPz｜CO｜cofun
Coprocessor z Operation	COPz cofun コプロセッサzはコプロセッサごとに定義されたオペレーションを実行する。CPUの状態はコプロセッサのオペレーションにより変更されない

命令	形式と説明　COPz｜BC｜br｜offset
Branch On Coprocessor z True	BCzT offset 遅延スロットの命令のアドレスに、16ビットのoffsetを2ビット左にシフトし、32ビットに符号拡張したものを加え、分岐アドレスを算出する。コプロセッサzの条件信号が真なら、分岐アドレスへ1命令遅れで分岐する
Branch On Coprocessor z False	BCzF offset 遅延スロットの命令のアドレスに、16ビットのoffsetを2ビット左にシフトし、32ビットに符号拡張したものを加え、分岐アドレスを算出する。コプロセッサzの条件信号が偽なら、分岐アドレスへ1命令遅れで分岐する

図10　コプロセッサ命令(MIPS Ⅲ)(1/2)

命令	形式と説明　COPz｜sub｜rt｜rd｜0
Doubleword Move To Coprocessor z	DMTCz rt, rd CPUの汎用レジスタrtの内容をコプロセッサzの汎用レジスタrdに転送する
Doubleword Move From Coprocessor z	DMFCz rt, rd コプロセッサzの汎用レジスタrdの内容をCPUの汎用レジスタrtに転送する

命令	形式と説明　op｜base｜rt｜offset
Load Doubleword To Coprocessor z	LDCz rt, offset(base) offsetを符号拡張し、レジスタbaseに加算してアドレスを生成する。アドレス指定されたダブルワードの内容をコプロセッサzの汎用レジスタrtとrt+1にロードする
Store Doubleword From Coprocessor z	SDCz rt, offset(base) offsetを符号拡張し、レジスタbaseに加算してアドレスを生成する。コプロセッサzの汎用レジスタrtとrt+1のダブルワードの内容をアドレス指定されたメモリ位置にストアする

図10　コプロセッサ命令(MIPS Ⅲ)(2/2)

命令	形式と説明　COPz｜BC｜br｜offset
Branch On Coprocessor z True Likely	BCzTL offset 遅延スロットの命令のアドレスに、16ビットのoffsetを2ビット左にシフトし、32ビットに符号拡張したものを加え、分岐アドレスを算出する。コプロセッサzの条件信号が真なら、分岐アドレスへ1命令遅れで分岐する。分岐条件が成立しなかった場合、分岐遅延スロット内の命令は破棄される
Branch On Coprocessor z False Likely	BCzFL offset 遅延スロットの命令のアドレスに、16ビットのoffsetを2ビット左にシフトし、32ビットに符号拡張したものを加え、分岐アドレスを算出する。コプロセッサzの条件信号が真なら、分岐アドレスへ1命令遅れで分岐する。分岐条件が成立しなかった場合、分岐遅延スロット内の命令は破棄される

図11 システム制御コプロセッサ命令(CP0) (2/2)

命令	形式と説明　COP0 | sub | rt | rd | 0
Move To System Control Coprocessor	MTC0 rt, rd CPUの汎用レジスタrtのワードの内容を、CP0の汎用レジスタrdにロードする
Move From System Control Coprocessor	MFC0 rt, rd CP0の汎用レジスタrdのワードの内容を、CPUの汎用レジスタrtにロードする
Doubleword Move To Coprocessor 0	DMTC0 rt, rd CPUの汎用レジスタrtのダブルワードの内容を、CP0の汎用レジスタrdにロードする
Doubleword Move From Coprocessor 0	DMFC0 rt, rd CP0の汎用レジスタrdのダブルワードの内容を、CPUの汎用レジスタrtにロードする

命令	形式と説明　COP0 | CO | func
Read Indexed TLB Entry	TLBR エントリHi、エントリLo0、エントリLo1、ページマスクレジスタにインデックスレジスタにより指示されているTLBエントリをロードする
Write Indexed TLB Entry	TLBWI インデックスレジスタにより指示されているTLBエントリに、エントリHi、エントリLo0、エントリLo1、ページマスクレジスタの内容をロードする
Write Random TLB Entry	TLBWR ランダムレジスタにより指示されているTLBエントリに、エントリHi、エントリLo0、エントリLo1、ページマスクレジスタの内容をロードする
Probe TLB For Matching Entry	TLBP インデックスレジスタに、エントリHIレジスタの内容と一致するTLBエントリのアドレスをロードする
Return From Exception	FRET(MIPS II) 例外、割り込み、エラートラップから復帰する
Return From Exception	RFE(MIPS I) 例外、割り込み、エラートラップから復帰する

命令	形式と説明　CACHE | base | op | offset
Cache Operation	Cache op, offset(base) 16ビットのoffsetを32ビットに符号拡張してレジスタbaseと加算し、仮想アドレスを生成する。仮想アドレスをTLBを用いて物理アドレスに変換し、そのアドレスに対し5ビットのサブオペコードで示されるキャッシュオペレーションを行う

図13 FPUの命令形式

●I-Type(イミディエート形式)

31　26	25　21	20　16	15　0
op	base	ft	offset

●R-Type(レジスタ形式)

31　26	25　21	20　16	15　11	10　6	5　0
COP1	fmt	ft	fs	fd	func

●その他

(1) 分岐

31　26	25　21	20　18	17	16	15　0
COP1	BC	cc	l	ft	offset

(2) CPU、FPU間データ転送

31　26	25　21	20　16	15　11	10　0
COP1	MT/MF	rt	fs	00000000000

op:6ビットのオペコード　COP1:6ビットのオペコード(0x11)
base:5ビットのベースレジスタ　fmt:5ビットのフォーマット
単精度(0x10)/倍精度(0x11)/32ビット整数(0x14)/64ビット整数(0x15)
ft:5ビットのソース(演算、ストア時)、またはデスティネーション(ロード時)
FPUレジスタ
fs:5ビットのソースFPUレジスタ　fd:5ビットのデスティネーションFPUレジスタ
offset:16ビットの符号付きオフセット　func:6ビットの関数指定
BC:5ビットの条件分岐を示すサブオペコード(0x08)
cc:3ビットのコンディションコード番号　l:1ビットのBranch Likelyの指定
tf:1ビットのtrue/falseの指定
MT:CPUレジスタrtの内容をFPUレジスタfsに転送
MF:FPUレジスタfsの内容をCPUレジスタrtに転送

図12 CPU命令のオペコードマップ

■Opcode

31-29 \ 28-26	0	1	2	3	4	5	6	7
0	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	COP0	COP1	COP2	COP1X	BEQL	BNEL	BLEZL	BGTZL
3	DADDI ε	DADDIU ε	LDL ε	LDR ε	*	JALX	MDMX	*
4	LB	LH	LWL	LW	LBU	LHU	LWR	LWU ε
5	SB	SH	SWL	SW	SDL ε	SDR ε	SWR	CACHE δ
6	LL	LWC1	LWC2	PREF	LLD ε	LDC1	LDC2	LD ε
7	SC	SWC1	SWC2	*	SCD ε	SDC1	SDC2	SD ε

■SPECIAL function

5-3 \ 2-0	0	1	2	3	4	5	6	7
0	SLL	MOV{T,F}	SRL	SRA	SLLV	*	SRLV	SRAV
1	JR	JALR	MOVZ	MOVN	SYSCALL	BREAK	*	SYNC
2	MFHI	MTHI	MFLO	MTLO	DSLLV ε	*	DSRLV ε	DSRAV ε
3	MULT	MULTU	DIV	DIVU	DMULT ε	DMULTU ε	DDIV ε	DDIVU ε
4	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	*	*	SLT	SLTU	DADD ε	DADDU ε	DSUB ε	DSUBU ε
6	TGE	TGEU	TLT	TLTU	TEQ	*	TNE	*
7	DSLL ε	*	DSRL ε	DSRL ε	DSLL32 ε	*	DSRL32 ε	DSRA32 ε

■REGIMM rt

20-19 \ 18-16	0	1	2	3	4	5	6	7
0	BLTZ	BGEZ	BLTZL	BGEZL	*	*	*	*
1	TGEI	TGEIU	TLTI	TITIU	TEQI	*	TNEI	*
2	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*	*	*	*
3	*	*	*	*	*	*	*	*

■COPz rs

5-3 \ 23-21	0	1	2	3	4	5	6	7
0	MF	DMF ε	CF	γ	MT	DMT ε	CT	γ
1	BC	γ	γ	γ	γ	γ	γ	γ
2	CO							
3								

■COPz rt

20-19 \ 18-16	0	1	2	3	4	5	6	7
0	BCF	BCT	BCFL	BCTL	γ	γ	γ	γ
1	γ	γ	γ	γ	γ	γ	γ	γ
2	γ	γ	γ	γ	γ	γ	γ	γ
3	γ	γ	γ	γ	γ	γ	γ	γ

■CP0 function

5-3 \ 2-0	0	1	2	3	4	5	6	7
0	ϕ	TLBR	TLBWI	ϕ	ϕ	ϕ	TLBWR	ϕ
1	TLBP	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
2	RFE	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
3	ERET χ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
4	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
5	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
6	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
7	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

MIPS IVで導入された。条件が真のときにFPUレジスタ間の転送を行う。条件の指定方式としては、CPUレジスタが0であるか0でないか、浮動小数点数の条件ビットが真(1)であるか偽(0)であるかの4通りがある。例としては、

```
movz.s  fp1,fp2,r3  // r3が0のとき転送
movn.s  fp1,fp2,r3  // r3が0でないとき転送
movt.s  fp1,fp2,cc3 // cc3が真のとき転送
movf.s  fp1,fp2,cc3 // cc3が偽のとき転送
```

などが考えられる。

④積和演算

MIPS IVでは浮動小数点演算の性能向上を目的としている。その最大の目玉はインデックス付きロード/ストア命令と積和演算である。積和演算は、MIPSには珍しく4オペランドを取る。通常のFPU命令ではフォーマット

図14　FPU命令オペコードマップ

■Opcode

31-29 \ 28-26	0	1	2	3	4	5	6	7
0								
1								
2		COP1		COP1X				
3								
4								
5								
6		LWC1				LDC1		
7		SWC1				SDC1		

■COP1 function

5-3 \ 2-0	0	1	2	3	4	5	6	7
0	ADD	SUB	MUL	DIV	SQRT	ABS	MOV	NEG
1	ROUND.L η	TRUNC.L η	CEIL.L η	FLOOR.L η	ROUND.W η	TRUNC.W η	CEIL.W η	FLOOR.W η
2	γ	MOV{T, F}	MOVZ	MOVN	γ	RECIP	RSQRT	γ
3	γ	γ	γ	γ	γ	γ	γ	γ
4	CVT.S	CVT.D	γ	γ	CVT.W	CVT.L η	γ	γ
5	γ	γ	γ	γ	γ	γ	γ	γ
6	C.F	C.UN	C.EQ	C.UEQ	C.OLT	C.ULT	C.OLE	C.ULE
7	C.SF	C.NGLE	C.SEQ	C.NGL	C.LT	C.NGE	C.LE	C.NGT

■sub

25-24 \ 23-21	0	1	2	3	4	5	6	7
0	MF	DMF η	CF	*	MT	DMT η	CT	*
1	BC	*	*	*	*	*	*	*
2	S	D	γ	γ	W	L η	γ	γ
3	γ	γ	γ	γ	γ	γ	γ	γ

■COP1X function

5-3 \ 2-0	0	1	2	3	4	5	6	7
0	LWXC1	LDXC1	γ	γ	γ	γ	γ	γ
1	SWXC1	SDXC1	γ	γ	γ	γ	γ	PREFX
2	γ	γ	γ	γ	γ	γ	γ	γ
3	γ	γ	γ	γ	γ	γ	γ	γ
4	MADD.S	MADD.D	γ	γ	γ	γ	γ	γ
5	MSUB.S	MSUB.D	γ	γ	γ	γ	γ	γ
6	NMADD.S	NMADD.D	γ	γ	γ	γ	γ	γ
7	NMSUB.S	NMSUB.D	γ	γ	γ	γ	γ	γ

■br

20-19 \ 18-16	0	1	2	3	4	5	6	7
0	BCF	BCT	BCFL	BCTL	*	*	*	*
1	*	*	*	*	*	*	*	*
2	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*

図15　FPU演算命令と動作

(1)COP1系列

func[5:0]	ニーモニック	動作
0	ADD.fmt	加算
1	SUB.fmt	減算
2	MUL.fmt	乗算
3	DIV.fmt	除算
4	SQRT.fmt	平方根
5	ABS.fmt	絶対値
6	MOV.fmt	転送
7	NEG.fmt	符号反転
8	ROUND.L.fmt	64ビット整数に変換。もっとも近い値への丸め
9	TRUNC.L.fmt	64ビット整数に変換。0方向の丸め
10	CEIL.L.fmt	64ビット整数に変換。+∞方向の丸め
11	FLOOR.L.fmt	64ビット整数に変換。-∞方向の丸め
12	ROUND.W.fmt	32ビット整数に変換。もっとも近い値への丸め
13	TRUNC.W.fmt	32ビット整数に変換。0方向の丸め
14	CEIL.W.fmt	32ビット整数に変換。+∞方向の丸め
15	FLOOR.W.fmt	32ビット整数に変換。-∞方向の丸め
16	—	予約
17	MOV{T,F}.fmt	条件転送。条件コードによって転送
18	MOVZ.fmt	条件転送。CPUレジスタが0のとき転送
19	MOVN.fmt	条件転送。CPUレジスタが0でないとき転送
20	—	予約
21	RECIP.fmt	逆数
22	RSQRT.fmt	平方根の逆数
23-31	—	予約
32	CVT.S.fmt	単精度浮動小数点数に変換
33	CVT.D.fmt	倍精度浮動小数点数に変換
34-35	—	予約
36	CVT.W.fmt	32ビット整数に変換
37	CVT.L.fmt	64ビット整数に変換
38-47	—	予約
48-63	C.cond.fmt	比較

fmt={S|D|W|L}

(2)COP1X系列

func[5:0]	ニーモニック	動作
0	LWXC1	インデックス付きロード。32ビット
1	LDXC1	インデックス付きロード。64ビット
2-8	—	予約
8	SWXC1	インデックス付きストア。32ビット
9	SDXC1	インデックス付きストア。64ビット
10-31	—	予約
32	MADD.S	乗算＋加算。単精度
33	MADD.D	乗算＋加算。倍精度
34-35	—	予約
36	MSUB.S	乗算＋減算。単精度
37	MSUB.D	乗算＋減算。倍精度
38-47	—	予約
48	NMADD.S	乗算＋加算＋符号反転。単精度
49	NMADD.D	乗算＋加算＋符号反転。倍精度
50-55	—	予約
56	NMSUB.S	乗算＋減算＋符号反転。単精度
57	NMSUB.D	乗算＋減算＋符号反転。倍精度
58-63	—	予約

図16　比較演算の条件と条件ビットにセットされる値

条件(cond)	関係				Unorderedのとき無効演算例外を発生するか
	>	<	=	Unordered	
F	0	0	0	0	しない
UN	0	0	0	1	
EQ	0	0	1	0	
UEQ	0	0	1	1	
OLT	0	1	0	0	
ULT	0	1	0	1	
OLE	0	1	1	0	
ULE	0	1	1	1	
SF	0	0	0	0	する
NGLE	0	0	0	1	
SEQ	0	0	1	0	
NGL	0	0	1	1	
LT	0	1	0	0	
NGE	0	1	0	1	
LE	0	1	1	0	
NGT	0	1	1	1	

(fmt)の部分がレジスタ指定(fr)となっているので，機能フィールド(func)でフォーマットまでを指定している。

```
madd.s   fd,fr,fs,ft
msub.s   fd,fr,fs,ft
```

は，それぞれ，

```
fd ← (fs×ft)+fr
```

および，

```
fd ← (fs×ft)-fr
```

という演算(単精度)を実行する。

おわりに

MIPSの命令セットアーキテクチャについて解説してきた。私見であるが，MIPSの命令セットはZ80に比べれば数段，MC68000と比べてもかなり洗練されていると思う。皆さんはどう思われたであろうか。これを機会にMIPS RISCに対する興味が深まってもらえれば幸いである。次回はMIPSの命令セット以外のアーキテクチャ(仮想記憶，キャッシュ，特権命令など)の解説を予定している。

Oh!X 質問箱

Q X68000EXPERTとSCSIのPDドライブと56kモデムをつなぎたいのですが，動かすにはどうすればよいのでしょうか。SxSIとかいうソフトですか？　古川卓也　佐賀県

A SCSIボードがあればPDの使用はほぼ問題ありません。SCSIボードがない場合は，おっしゃるようにSxSIドライバを導入するのが有効でしょう。現状ではX68000用のSCSIボードを入手するのは困難になってきていますので。X68000のハードディスクインタフェイスはSASIとSCSIの中間的なもの(むしろSCSIをモディファイしたもの)で，コネクタなどはSCSIと共通のものが使われています。信号などを制御して標準的なSCSIと同じ動作にしてやろうというのが，SxSIドライバの骨子ですが，パリティ関係など，一部ハードウェア改造することを推奨されている部分もあります。詳しくはドライバを入手したときについてくるドキュメントをお読みください。

ポートがSCSI化されることでのデメリットはほとんどありません。ストレージ関係などの一般的なSCSIデバイスはHuman68kでそのまま扱ってもほぼ問題ありません。MOなどはIBMフォーマットで使用するためのドライバなどが出回っています(p.262参照)。

古川さんはインターネット接続環境はお持ちのようですので，ご自分でいろいろ探してみられることをおすすめします。ソフトウェアを入手するということだけなら，そのほうが手間がかかるのですが，いろいろ関連情報が入ってきますので(しかし，最近はリンク切れページも多いなあ……)。

なお，56kモデムについては，どの程度使えるのか微妙なところです。最近のモデムは，パソコン側のドライバでプロトコル処理を行い，モデム本体は信号の変換しかしないなどといったものも増えていますので，以前のようにモデムだから機種を問わずに使えるという状況ではなくなってきています。なにぶん，機種が無数にあるので，購入前にできるだけ確認を取るのがよいでしょう。最低でもヘイズATコマンドで動いてくれなければ使いものになりません。加えていえば，通常のX68000のシリアルポートを使っていると，56kモデムはたとえ動作したとしても無駄なのかもしれません。

Q インターネットでWebページを見ていると，たまにスクリプトエラーとかで止まってしまうことがあるのですが，これは元のページがちゃんとしたスクリプトを書いてないということでしょうか。ブラウズにはWindows98のIE4.01を使っています。ブラウザごとにJavaScriptの文法とかが違うというのは理解しています。しかし，昨今ではIE4で動作確認しないWebページというのも，そう多くはないと思うのですが。これはほかに理由があるものなのでしょうか。　保木茂人　神奈川県

A (ああ，なんかジオの掲示板みたいだ)。ひとり言です。スクリプトエラーが発生する場合に，比較的多く見られるのが，スクリプト中で日本語文字列を使っている場合に，文字化けが起こると，ほかの文字を巻き込んでしまうのでエラーが発生するという奴ですね。この場合，ブラウザの文字コード設定を修正すれば解決します。

次に多いのが，スクリプト側がハンドシェイクをきちんとしていないので，回線状態が悪くなるとタイムアウトでエラーが出るというものです。この場合，作者がローカル環境などの「非常に回線状態がいい」状態でしかデバッグしてないと多発することがあります。対処法としては，リロードを繰り返すとか，エラーを無視していればちゃんと進む場合が多いですね。サーバとブラウザとのやり取りでは，各部できちんとしたハンドシェイクを行おうとすると，思いのほか厳重にコードを書かなければならないみたいです。それでもなお回線状態によっては，うまく動かないというケースもあるようで。

そのほかには，フレームページで親フレームと通信してるのに，子フレームを直にリンクするとエラーが出るとかですか。これも無視すればたいてい大丈夫なはずです。

スクリプト関係はブラウザごとにローカルな差異が多く，完全な互換を取るのは難しいようです。復刊号のJavaScriptの記事，および付録CD-ROMのなかに古籏氏の作った一覧表が入っていますので参照するとよいでしょう（今号の付録ディスクのJavaScriptのフォルダにも丸ごと入っています)。

Q Windowsの表示系はどうしてY軸が下からになっているのでしょうか。不便でたまりません。Windowsを作った人は，なにかグラフィック端末に触ったことはなかったのでしょうか。それと，そもそも普通のグラフィック表示系ではなぜ画面左上を原点としているのでしょうか。　島本吉弘　新潟県

A Windowsの場合は，おそらくグラフの第1象限を画面とするのが美しいと判断したのでしょう。グラフが書きやすいとか(大差ないか)。誰がどう決めたのかなどはよくわかりません。確かに，ほかの環境とはまったく違うのでグラフィックファイルビュアなどでは困りものですよね。データフォーマットまでそうすることはないんじゃないかという気はしますが「ビットマップを下から描く」のを仕様決定者は自然だと思ったのでしょう。

それ以外のごく普通のグラフィック座標については，おそらくキャラクタのカラム/ラインの関係をそのまま延長したことによるのだと思われます。画面の走査線順にディスプレイ座標を決めたということも考えられなくはないですが，もともとが文字文化ですから，文字座標と同じ方向にするのが自然だったのでしょう。

ちなみに，以前雑誌のインタビューでビクターの技術者(テレビを初めて作った人に)に「なぜ左上からスキャンするのか」と聞いていたものを見たことがありますが，開発者自身もそういうことをまったく意識したことがなかったようで(ほかの方向からでも別に支障はなかったという事実を)，おそらく文字を書く流れなどからそうスキャンするのが自然だと思い込んでいたのでしょう。右利きの人がなにかを塗りつぶすときにはたいてい，こうスキャンすると思います。とにかく，たまたま最初の試作機でそうしていたので，以降，世界中のCRTはそうスキャンするようになったそうで。どっちが自然な流れかというのは，人によって見解が違うでしょうが，普通は上からだよなあと私も思います。

質問募集

このコーナーでは皆様からの各種質問事項を募集しています。ただし，掲載の関係上，緊急を要するようなトラブルに対する質問には向いておりません。日頃疑問に思っていること，知識として知りたいことなどをお寄せください。なお，具体的な問題の場合は，機種名，OS名，アプリケーションの場合はバージョン番号などを明記しておいてください。

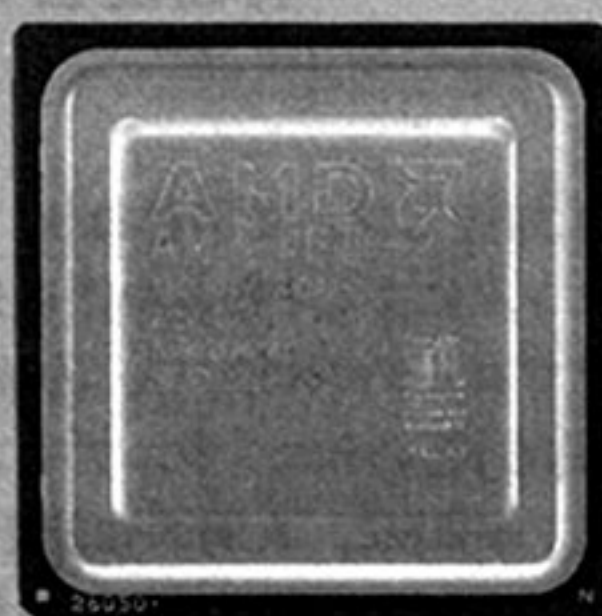
俺はC++から始めてやるぜ!!
Microsoft Visual C++ 6.0 Standard Edition
マイクロソフト
☎03-5454-2300

B 2名

■モニター応募要項

■応募資格：自分はクリエイターだと自負する方，もしくはそれを目指している方。本誌綴じ込みの愛読者カードに必要事項（住所，郵便番号，氏名，年齢，職業，連絡先，モニターの希望番号と商品名）を明記のうえ，ご応募ください。応募の締め切りは'99年11月1日（当日消印有効）です。
■モニター期間：商品受領より3か月。商品によって発送に日数がかかることをご了承ください。
■モニター期間中に最低数回のモニターレポートを提出していただきます。レポートは書面またはテキストファイルの入ったフロッピーでDOS/V magazine内Oh!X編集室モニター係までお送りください。なお，レポートの返却はいたしません。
■該当者の発表は発送をもって代えさせていただきます。

A 2名

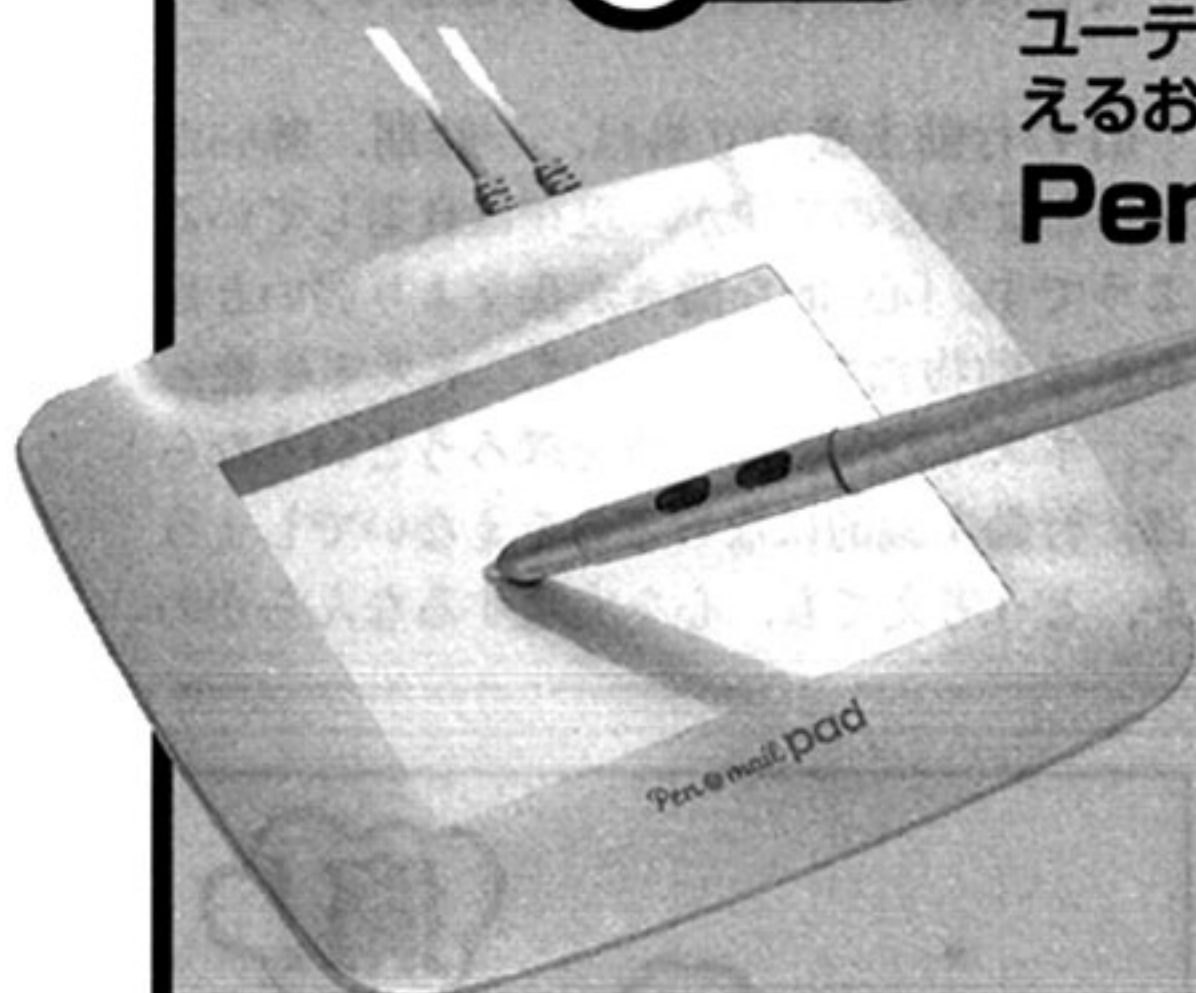
まずはVBから始めよう!!
Microsoft Visual Basic 6.0 Learning Edition
マイクロソフト
☎03-5454-2300

あたしはPCを組み立てることから始めるわ!!
AMD K6-2/350MHz
日本AMD
http://www.amd.com/japan/

C 1名

Oh!X '99年夏号 モニター募集 愛読者プレゼント

1 2名

ユーティリティソフトと一緒に使えるお手軽タブレット
Pen@Mail JOY
ネオスコーポレーション
☎03-3353-1621

3 3名

ライター西川善司さんに勝ちたいあなたは
ポピュラス ザ・ビギニング 公式ガイド
ソフトバンク パブリッシング
☎03-5642-8100

非英語圏のWebサーフィンもばっちり
internet Surfing PACK 1
ダイナラブ・ジャパン
☎03-3224-2170

2 2名

■プレゼント応募要項

■本誌綴じ込みの愛読者カードに必要事項（住所，郵便番号，氏名，年齢，職業，連絡先，プレゼントの希望番号と商品名）を明記のうえ，ご応募ください。応募の締め切りは'99年11月1日（当日消印有効）です。
■当選者の発表は発送をもって代えさせていただきます。

注）記入もれ，および希望商品が複数記入されている場合は無効にさせていただきますので，ご了承ください。また，商品の都合上，希望されたものと違う商品が当選する場合があります。
注）雑誌公正競争規約の定めにより，プレゼント該当者は本号の，ほかのプレゼントには当選できない場合があります。

知能機械概論

Tinoukikai-Gairon

第102回

「計算機の中の心/命」について考える

有田隆也 Arita Takaya

究極的なアイデア

計算機を限界まで使いこなすといったいどんなことまでできてしまうのか？と思います。このことをじっくりと静かに考えようと思う一方で，我々の時代は，さまざまなかたちでその答えを思いもかけないような新技術，あるいは新製品として次から次へと我々に与えてくれます。社会の情報化はSF小説以上のスピードで進み続けているとでもいったらよいのでしょうか。

私は，計算機科学と社会の関係の未来に関する多くのことについては楽観的でありまして，SF作家オーウェルが小説「1984」で書いたような暗い社会がやってくるというよりは，このような情報テクノロジーの想像を絶する発展がもたらすだろうさまざまな結果に関して，脳天気な期待を持っています。2人のスティーブがガレージの中で名機Appleを作って世の中に送り出したときの彼らの胸中といった感じでしょうか(いまなお)。

評論家的な未来予測の話は横に置いて，もっと究極的なことを考えたいと思います。それは，高性能なハードウェアを用意して，計算機のプログラムだけで「心」や「命」ができてしまうか？ということです。この質問に対して，早々と私の考えをいってしまうと，「心も命も計算機の中に実現できてしまう！」です。

そのように思っているだけでなくて，誰もが納得するような，計算機の中の心や命というものを実際に見せることができればよいのですが，それはさすがに現時点では無理です。そういうわけで，このような主張の背景にある基本的な考え方というものを述べたいと思います。

心と脳はどんな関係か？

「生命」の前に「心」からいきましょう。計算機の中の心という問題に関しては，人工知能の研究領域でさまざまに議論されてきましたので，その成果をごく簡単に紹介したいと思います。

人工知能とは，文字どおり，計算機のプログラミングによって知能を作り上げてやろうというアプローチですが，人工知能には，そのような「作る」というエンジニアリングの側面だけでなく，同時に，人間の知能を「解明する」というサイエンスの側面もあります(この側面は，認知科学とか，認知心理学などの領域ともオーバーラップしてきます)。知能といっても，情報処理的な意味における知能だけでなく，「心」なる謎をも意味すると考えていいようです。ただし，心理学などの研究領域が心にアプローチするのと比べて大きく異なるのは，人間の知能や心の具体的なメカニズムを解明していくことを直接的には対象にせずに，コンピュータのプログラムによって，その働きや機能としては，人間の知能や心に共通するところを実現しようとする点です。

心をコンピュータのプログラムで作ってしまうという，一見荒唐無稽な話ですが，これが可能かどうかの議論を意義あるものとするために，もう少し「心」とはいかなるものかということを考えてみましょう。ひとつ，具体的に問題を設定します。それは，「心と身体(脳)はどのような関係にあるのか？」です。この問題は，「心身問題」と呼ばれてきました(なんだか病気みたいですけれど)。

このとても難しい問題に答える立場として，まず，次のように大きく2つに分けることができましょう。

1) 一元論

この世にあるのは身体(物質)だけである(唯物論，あるいは物理主義)。

2) 二元論

心と身体はそれぞれ独立に存在しうる別種の実体である。

心という，物質では説明できないなにかが，物質とは関係なく存在するという2)の主張は，宗教的な場面や芸術的な文脈において，あるいは，人間の素朴な感情にとって，自然なもののように思われます。しかし，ここでは，この立場は検討しないことにして，先を急ぐことにしましょう。むろん，2)が間違っていると証明することはできないでしょう(間違っていると私は思ってはいますけれど)。

さて，1)の立場，つまり，心というなにか独立したものが存在しないとすると，「では，心と呼ばれているものはいったいなんなのだ？」という疑問に答える必要が生じます。それに対しては，次の3つの代表的な立場があるといえます。わかりやすくするために，それぞれの立場に関して，「痛い！と感じているという心の状態はなにか？」という具体的な例を使って説明しましょう。

a) 行動主義

「痛みを感じている」と形容される心の状態などない。「傷口を押さえて，額に汗をかきながら，『痛っ！』と叫び，……」という行動の記述の省略形である。

b) 同一説

「痛みを感じている」と形容される心の状態とは，脳の中の神経回路網において生じるある興奮パターンという脳内における物理的な出来事にほかならない。

c) 機能主義

「痛みを感じている」と形容される心の状態とは，我々においては，脳というハードウェアにおいて実行されているプログラムのある状態のことである。脳以外のハードウェア(たとえばコンピュータ)を使ってもそこにおいて心というものはありうる。

a)の行動主義の立場は，一時期，勢いがあったようなのですが，現在は衰退しているようです。「心」的な働きが我々より弱いと思われる動物たち，たとえば，ネズミを使って，ネズミの内的な状態を探ろうというならば，行動主義的にならざるをえないでしょうが，どう考えても，心を実現するなんらかの

状態はあると考えざるをえないのではないのでしょうか。

b)の同一説のほうは，客観的に見て，かなり有力だと思われます。今日，脳の働きが次から次と明らかにされ，心というものの実体が明らかにされつつあります。ただ，ここで，問題なのは，確かに我々においては「心」は脳内の出来事であるが，その出来事というものは，いってみれば，ソフトウェアによる計算状態のことであって，だとすれば別のハードウェアを想定すれば，そこでも実現可能かどうかということです。もし，実現可能ならば，これは，c)の機能主義の立場に相当することになります。

このことは，「心の多重実現可能性のテーゼ」と呼ばれるそうです。ソフトウェア/ハードウェアというと，すでに計算機の世界の話ですが，もちろん，多重実現可能性とは，計算機だけで実行できるという話ではありません。たとえば，ネズミを捕まえる装置を考えましょう。バネで挟む仕掛けとチーズで作ることもできるでしょうし，あるいは，感電する仕掛けと化学的な誘導剤とで作ることもできるでしょう。この場合，ネズミを捕まえるという「機能」は同一であっても，メカニズムや材料は異なって実現できるということです。これが，多重実現可能性です。心もそうでしょうか?

人工知能の研究にとっては，機能主義の立場がもっともしっくりきます(私にとってもしっくりきます)。心の機能をコンピュータで実現しようということですから。ただし，すべての人工知能の研究者がこの立場をとっているということではありません(そもそも，この問題に関与する必要のない研究をしている人工知能研究者が少なくないようなので)。しかし，人工知能研究の土台を支える考え方としては，機能主義的傾向が強いということはできると思います。

ところで，現在はやっていませんが，以前，ある授業で心身問題に関して上記のような説明を簡単に行ったあと，自分がどの立場に近いと感じるかというアンケートをとっていたことがあります。それぞれの学生は，100%を，たとえば，機能主義60%，同一説40%などのように割り振って答えます。その結果は，予想どおり，同一説がかなり強いという傾向が見られます。また，機能主義もそこそこ支持を得ます。しかし，驚かされたのは，別のことです。

上記の最初の分類のところで，実はひとつの分類が省略されています。唯物論，二元論があるのならば，当然，唯心論という立場も論理的にはありえます。「この世はすべて無だ」という，仏教的な思想とでもいいましょうか，物質そのもの存在を認めないという過激な立場です。このような立場に関しても，一応ざっと説明します。そして，結果として，このような立場が確実に支持を集めており，数十パーセントもの賛同を集めることもあったのです(そんなもんかねー)。

命と体はどんな関係か?

さて，今度は「計算機の中の命」ということを考えてみることにしましょう。心というものが，ソフトウェア的なものとするならば，計算機の中に実現するということもいくぶん説得力がありそうな気がしますが，今度は命です。命，生命というものがソフトウェアだといっても皆さんはすぐには理解してくれないように思われます。

バイオなどのテクノロジーを使って，モノとしての生命を作り出すというのならば，なんとなく現実味があるような気もします。一方，モノとしての心をなにかの物質で作るということになると，逆に難しそうに思われます。まとめると，
1)なにか，目に見えるモノとして作るならば，生命のほうが何となく実現性が高く感じられるが，
2)計算機の中に実現するとなると心のほうが実現できそうに感じられる
ということです。

たぶん，モノとして作るならば，進化の過程がそうであるように心というものは，生命よりはあとにできあがる難しいものであるということが，1)の理由でしょう。一方，2)の理由は，心がソフトウェアであるという形容が，命がソフトウェアであるということより，現時点では説得力があることによるのでしょう。

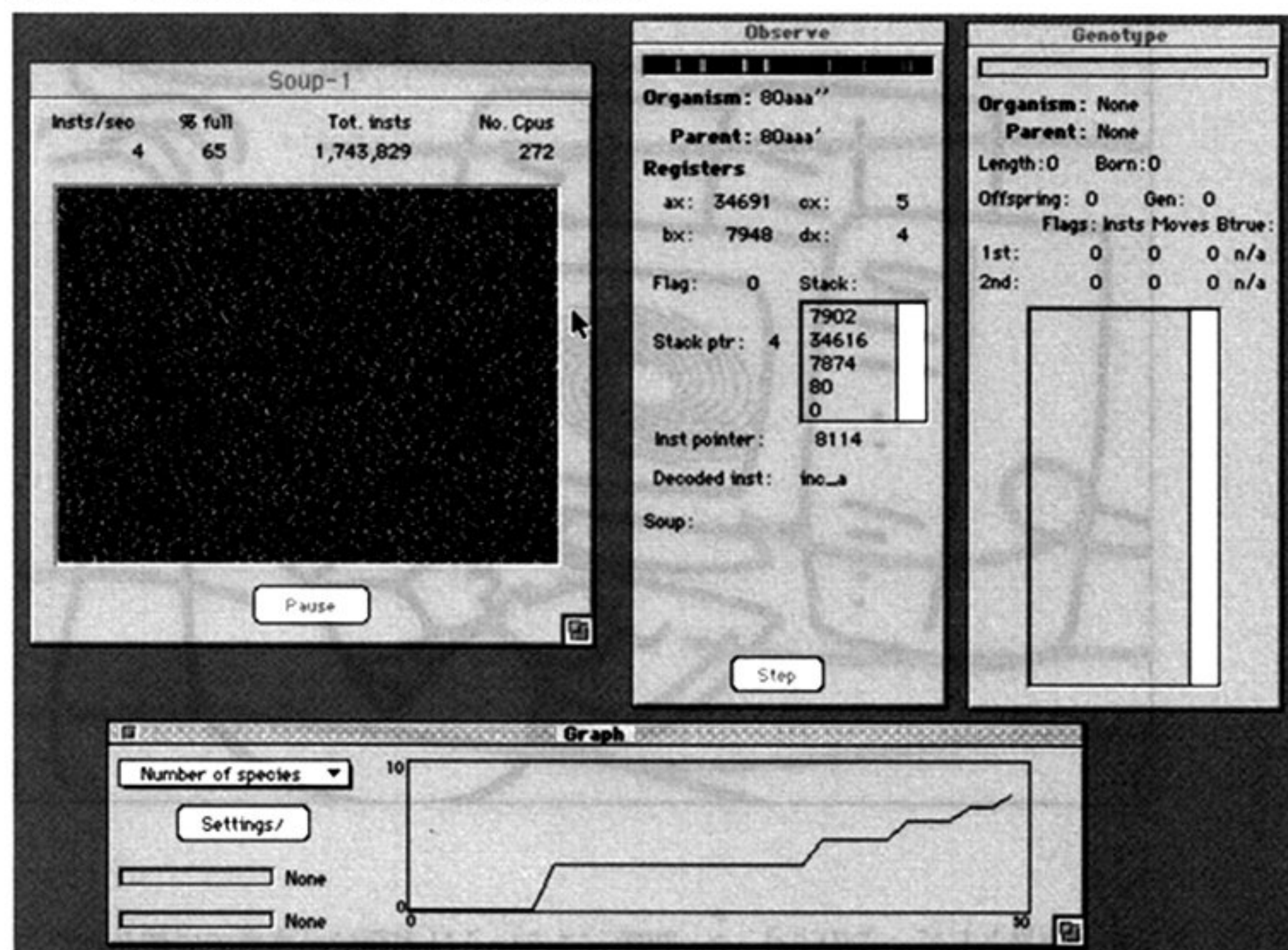
ここで，命に関しても，心身問題の議論によって明らかになった上記のような立場で分けて考えるとどういうふうになるかやってみることにしましょう。今度考えるべき問題は，「命身問題」(こういう日本語はいままでに使われたことがないような気がする……)ということになります。生命は身体とどのような関係にあるのか? という問題です。

心身問題の場合と同様に，まず次のように二分することができます。

1) 一元論
この世にあるのは身体(物質)だけである(唯物論，あるいは物理主義)。
2) 二元論
生命と身体はそれぞれ独立に存在しうる別種の実体である。

この2)に関しては，命の実体として「生気」というものを考える生気論というものがあります。このあたりは，話としては面白い面もあるかもしれませんが，オゾマシイ世界にのめり込みそうなので軽く流すとことにしまして，1)のほうをさらに考えることにします。心身問題と同様に3つの立場に従います。

a)行動主義
生命とは「歩いて，食事をして，子どもを作り……」などという振る舞い全体に対して名づけるべきものであり，システムへの入力と出力に対するある種の記述である。
b)同一説
「生きている」と形容される状態とは，我々が生物と呼んでいる有機物から構成される物理システムにおいて起こる出来事の総称にほかならない。
c)機能主義
「生きている」と形容される状態とは，我々が生物と呼んでいる有機物から構成される物理システム上で実現されているある種の高次の状態であり，そのような物理システム以外のハードウェアにおいても，

図1　Tierraシミュレータの画面写真

原理的に実現可能である。

「生命」という概念が「心」の概念より，具体的であるため，生々しい「生命」の概念をブラックボックスに閉じ込めるというa)の立場は，心身問題の場合に比べて必然的に説得力がないものになっていると思います。b)は，人間を含む生物の遺伝子に関する情報のメカニズムや，生命に関するもろもろの謎が解き明かされつつある今日において，ますます強い立場になってきているように思われます。

さて，c)の立場は，「生命の多重実現可能性のテーゼ」を支持する立場そのものです。「我々の知っている生物の体とは別のハードウェアで生命を実現する」といっても，想定するハードウェアにもいろいろあります。たとえば，いわゆるバイオ技術を使って人工的に生物を(クローンとかではなくて，人工の物質で)作り出すということも意味しますし，また，計算機でなんらかのプログラムを実行させるだけで，その実行状態が生命に相当すると主張する(Tierraモデルなど)ことも意味します。

知能や心を計算機内に実現しようとする人工知能研究が「心身問題」において機能主義の立場に近いということとまったく同様の関係で，生命現象を計算機内で実現しようとする人工生命研究は「命身問題」において機能主義の立場に近いということができます。当然，私は，機能主義の見解を支持します。

「生命の多重実現性」こそ人工生命の核心

人工知能研究者のすべてが心を計算機のプログラムで実現できると考えているわけではないのと同様に，人工生命研究者のすべてが生命を計算機のプログラムで実現できると考えているわけではありません。

その原因として，人工生命研究者といっても，研究の方法論，問題意識，動的システム観といったものこそ同じでも，生命システムを研究している人ばかりではないということがもちろん挙げられます。たとえば，私が人工生命の研究として言語の進化モデルを計算機で実験して研究している際に，命身問題に対してどういう立場をとるかということが直接的に問われたり，考えなくてはならないということはありません。

一方，生命現象そのものに直接関連する人工生命研究を行う場合，この命身問題は実はとても重要な問題となり，次のどちらの立場をとるのか迫ってくることになります(研究者本人にその問題の自覚はなくても，どういう観点から研究しているかで大きく異なってくるということです)。

1)弱い人工生命の立場
命の多重実現性を認めない立場。計算機の中に実現されているものは，現実のなにかを「シミュレート」したものである。そこで，生命に関する本質的な現象そのものが本当に起きているわけではない。

2)強い人工生命の立場
命の多重実現性を認める立場。計算機の中で起こっていることは，現実のなにかをシミュレートしたものではなく，生命に関する本質を実現するリアルな現象である。

弱い人工生命の立場は，台風のシミュレーションというものを考えるとわかりやすいと思います。ある人が台風のシミュレーションをするために，ミクロな記述から台風のようなマクロな挙動が計算機で再現されるようなとても巧妙なプログラムを作るのに成功したとします。

このとき，計算機の中で起こっているのは，台風のシミュレーションであって，台風そのもののわけがないでしょう，というわけです。これと同様に，どんなに巧妙なプログラムで生命現象を作り出したといっても，それが生命そのもののリアリティになるわけがないというのが弱い人工生命の立場です。

一方，強い人工生命の立場を認めさせるもっとも強力な方法は，誰もが「これは生きてるっ!」と唸らせるようなプログラムを作って，実際に見せることです。とはいっても，それは人工生命研究の究極的な目標ですから，現時点ではできるわけがありません。しかし，それに近いものとして，もっとも有名なのは，本連載でも取り上げたことのあるTierra(ティエラ)モデル(図1)です。

命身問題の観点からTierraを見ると，重要なことは，「現実世界のなにか具体的なものごとがプログラムの実行によってシミュレートされているわけではない」ということです。そこには，単に自分自身をコピーするという処理が実現されるような機械語プログラムが走っていて，そのプログラム同士の相互作用によって(パターンマッチによるアドレッシング，巧妙な命令セット，ほかのプログラム領域内の命令の自由な実行などの仕掛けも手伝って)，寄生や重寄生などの興味深いさまざまな現象が起こります。確かになにかのシミュレーションとは呼べないでしょう。

私は，強い人工生命の立場に立つことによってこそ，面白くてわくわくするような研究が可能になると思っています。理由を以下に書きます。なんらかの現実世界の生命現象のシミュレーションとしての人工生命は，実証科学として，現実との照合を絶えず求められることになります。その結果，さまざまな現実の美しくない部分も忠実にシミュレートしなくてはならなくなり，普遍的な真理にたどりつきにくくなるのではないかと思うのです。その結果，従来の見識にとらわれない新たな生命観へたどりつくのが難しくなるのではないでしょうか?　人工生命の真の創始者と考えられるフォン・ノイマンという天才研究者は，生命現象のひとつの本質である自己増殖というテーマに取り組み，自己複製が可能なマシンをセルオートマトン上に(パターンとして)作ることが可能なことを示しました。そのマシンは，自分自身に関する情報を記述したテープと，万能チューリングマシンと，万能製作マシン(図2)から構成されています。このテープは，自分自身の構造を作っていく際に頻繁にアクセスされ，また，子機械のためにテープ自身を複製する際にもアクセスされます。

このような構造は，染色体に基づく生命の

遺伝の構造に本質的に極めて近いものといえます。ここで，重要なことは，フォン・ノイマンは，生命の遺伝のメカニズムをシミュレートして，このようなマシンの自己複製のモデルを作り上げたのではないということです。なぜならば，そのような生命の遺伝子レベルのメカニズムの発見は，当時，まだなされていなかったのです！

要するに，フォン・ノイマンは，実際の生命現象を直接シミュレーションしたのではなく，その現象を深く考え，本質を抽出してモデルをリアルに作り上げる方法によって，結果的には，生命現象の謎の解明に先に到達できたといえるのです。

食べることを知らない生物なんて

そういうことで，強い人工生命というものは，命身問題における機能主義の立場を土台にすることにより，(うまくいけばの話ですが)生命に関する新しい知見，あるいは，新しい生命観に到達できるのだということを主張したつもりなのですけれど，このような主張に対する反論はもちろんありえます。

もっとも手ごわい反論は，次のようなものでしょう。

「確かに，Tierraモデルにおける，自分自身を複製するというプロセスは，生命に関する本質的なプロセスである自己増殖のプロセスを実現していると考えることができるだろう。しかし，それは自己増殖というプロセスがもともと計算論的な性格を持っていたためにすぎない。生命に関するほかの本質的なプロセス，たとえば，消化(あるいは新陳代謝や捕食などのエネルギーに関するプロセス)は計算論的ではない。このようなプロセスは計算機内では決して実現されえないだろう。」[文献1]。

このような主張にどのように反論しましょうか？先に述べたように，強い人工生命に値する実物を見せる研究を着々と進めるというのが，ひとつのやり方でしょうが，無理だといわれているのですから，現時点での反撃として，次のような回答を用意します。

1)計算機でプログラムを実行する際，計算機のハードウェアにおいて，電力の供給，消費などが実際に行われており，そのことに着目して説明する。
2)エネルギーという内部状態変数を作り，明示的に操作する。
3)モデル内において，エネルギーに対応する概念を設定したり，解釈して，対応づける。
4)エネルギーの概念の現れない人工世界を対象とする。
5)エネルギーは生命にとって本質ではないとする。

図2 フォン・ノイマンの万能製作マシン

1)はやや苦しい感じです。というか，せっかくのモデルの世界に現実世界の概念を直接組み込むので，美しくないという感じです。2)はありえそうな感じもしますが，このようにすると，通常のシミュレーションっぽくなり，先に主張していた強い人工生命から後退してしまうので避けたいところです。

3)はうまくいけばもっとも自然な説明ではないでしょうか。太陽エネルギー，資源，餌を競い合って奪いあうのが現実世界ですが，たとえば，Tierraの人工世界では，CPUの割り当て時間やメモリ領域をプログラムたちが取り合って進化していきます。そのような関係をきっちりと整理して説明すればよいのです。

4)は3)がうまくいかない場合にはしかたがなく取るというような態度だと思います。

5)に至っては，開き直りに相当する態度ですね(力まかせに，この路線で突っ走っていくことも，ひとつのやり方とはいえますが)。

結局，生命そのものを対象とする場合にとどまらず，社会や経済や言語などを対象とする場合でもそうですが，強い人工生命の立場で研究を行う場合，重要なことは，次のようなことだと思います。

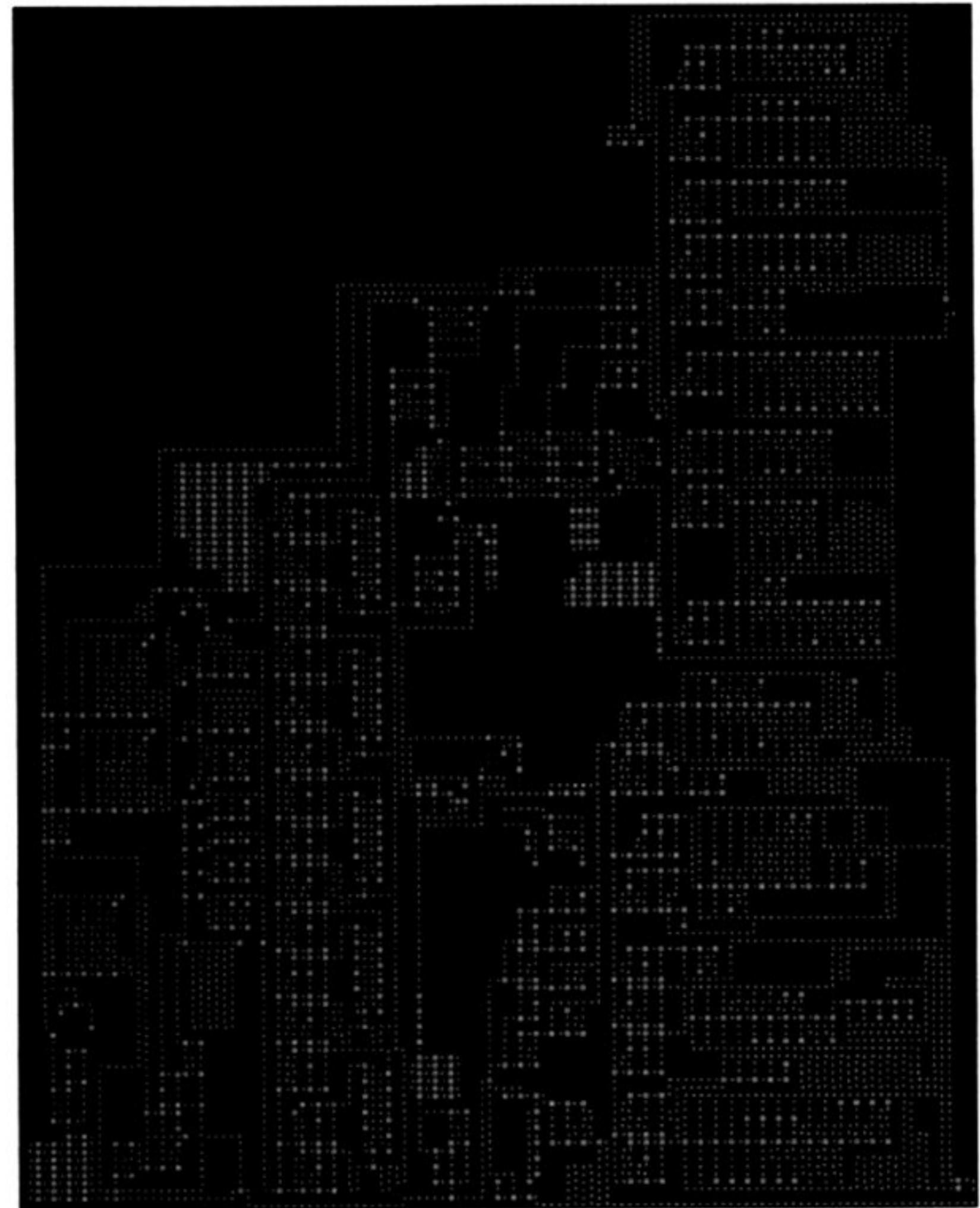
1)対象とするシステムの本質を捉えたモデル化を行う。些細とみなせる部分は積極的に切り捨てていく。あまり現実世界の細かな部分を拾っていくと，本質は隠され，普遍性が低くなっていくからである。
2)とはいっても，現実世界と人工世界の対応づけには絶えず注意すること。ひとりで，怪しげなプログラムを作って喜んでいるだけでよいのならばいいが，やはり，普遍的な知見とするには，現実世界との関係を意識し続けること。

もちろん，もっとも重要なことは，

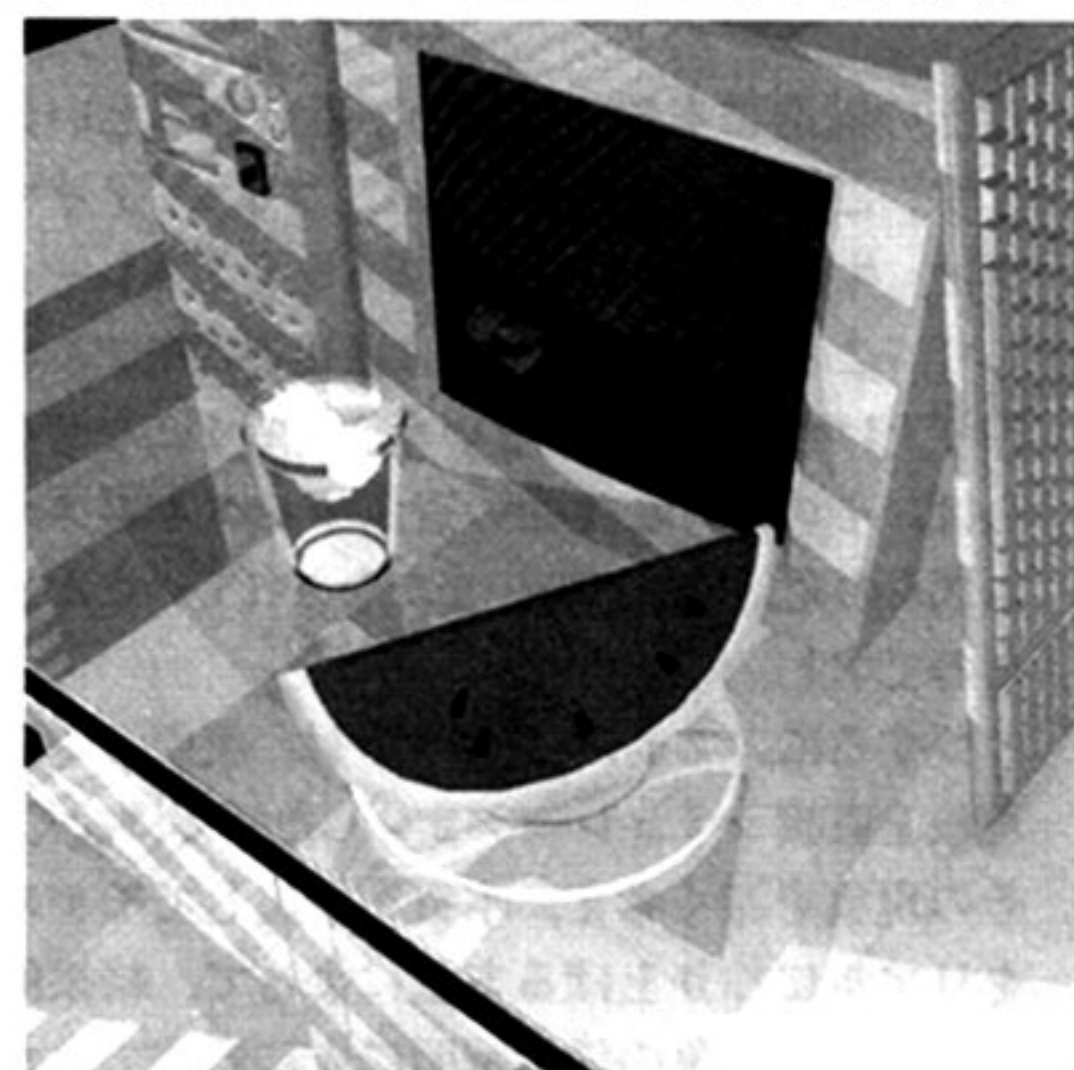
3)面白い計算機実験の結果をひねり出すこと。

です。

でも，なにが面白いかといわれるとまた難しい話になりそうですが，とりあえず，プログラムの文面を完全に理解しても，(通常の頭脳の程度では)どうしてこういう結果が出たのかわからないような結果のことといっておきましょう。ひとことで言うならば，創発性が高いということです。

参考文献)
[1]E. Sober, "Learning from Functionalism - Prospects for Strong Artificial Life", Artificial Life II, pp. 749－765, 1992.
[2]有田隆也，「人工生命」，科学技術出版。
2年かかってやっと書き終わりました。第1章で命身問題にも触れています。
私のホームページに紹介があります。
http://www2.create.human.nagoya-u.ac.jp/~ari/
[3]フォン・ノイマンの万能製作マシンのシミュレータ
http://alife.santafe.edu/alife/topics/jvn/jvn.html

STUDIO X ON AIR

FROM READERS TO THE EDITOR

初夏号の予定が夏号になったものの，なんとか夏のうちにはお届けできました（よね？）。それではさっそく春号の読者の皆さんからお寄せいただいたパワフルなご意見を紹介していきましょう。

◆年に一度だけナイコンが活躍できる場である「言わせてくれなくちゃだワ！」はいつやりますか？

畔地真太郎（29）兵庫県

別にちゃだワでなくても活躍できますよ。なんなら毎回やってもいいんですが，ハガキの量がもうちょっとほしい感じ……でしょうか。

◆Tcl/Tkって本当に「ティクルティケー」と読むんですか。

石門孝市（24）長崎県

チックル/チーコと読みます（嘘）。

◆X68000のことを過去のものなんていわないでくださいよ。まだ楽しみ尽くしてないのに。旧Oh!Xが休刊してからX68000に出合った人も大勢いるのだし。

鵜田幹人（21）愛知県

現役ユーザーの方には失礼しました。X68000の範囲でできることもまだまだあるわけですが，前に進んでいくのはなかなか難しいところです。一度休刊して復刊したのはその辺を整理して再出発という意味がありましたし。走り出したからには今後も前に進んでいきます。

◆電子メールがこれだけ広がっている現在，こうして読者ハガキをシコシコと書いていると「あ～，こんなところまで昔のままなんだ」と思う。

木村亮（25）東京都

いえ，別に電子メールでもOKなんですけど（打ち込まなくていいんでこっちも楽だし……）。

◆手に取ってあまりの厚さ（重さ）に驚き，内容の濃さにもっと驚き。これほどの内容についていくには読者も相当の修業が必要ですね。読んでいて思ったのは「旧Oh!Xも機種別枠を取っ払ってこういうのをやってたら休刊にはならなかったろうになあ」ということでした。いまはX68000しか持っていないのですが，いつかほかのマシンを購入したら貴誌を参考に思いっ切り遊んでみようと思います。

金城孝（23）鹿児島県

逆にいえば，当時の状況で機種別枠を取り外すには，休刊が必要だったということでしょう。単に枠をなくす方針にしただけで記事ができるものではありませんし，準備期間にずいぶんかかってしまったという感じです。

◆復刊号のような新鮮な感覚が少なかったように思われる。私個人がゲームに関心がないせいかもしれないが，同じゲームの記事にしてもOh!X全盛時のほうがインパクトがあったように思う。かつてはマシン語をはじめ，ハード系に触れた記事が多かったと思う。たとえMacやWindowsの記事が中心となり，マシン語は減らさざるをえないにしてもハード系を扱い続けてほしい。かつてのOh!Xはその方面ではかなりよい先生であったから。

五十嵐豊（32）神奈川県

以前もそうでしたが，毎回扱う内容に変動はあります。ハードウェアをやらないというわけではありません。むしろ強化方向で考えています。ただ，今回のハード系の記事はもともと春号に入るはずだったものでしたけど，なかなか捌ききれません。

◆PS，DC，GB，WS，SSと家庭用の開発環境を発売してもらいたいゲーム機はたくさんありますが，そのなかでもいちばんほしいのはWSでしょうか。白黒液晶なのでプログラマでも絵が（ほかと比べて）楽に作れて，音もよい。となると，アイデア勝負。ゲームの原点に立ち返れる気がします。

大矢裕（25）北海道

WonderSwanですか。開発環境を一般に公開するというので，問い合わせはしたのですが，なんとも要領を得ない感じでした。新聞のインタビューでやりたいねといったら大きく載っちゃってどうしよう状態なのかもしれません。公開されるようだったら，当然扱いますのでご心配なく。

◆パソコンの話になるとスペックや技術のことになりがちであまり面白くない。価格対性能比が急激に向上したパソコンでいったいなにをしているのかが気になりますが，ワープロや表計算だけならPentium IIIなんていらないと思う。目的をはっきりさせないとダメだよね。

西谷一獲（23）愛知県

最近のハイスペック機の目的といえば，やはりゲームでしょう。それとベンチマークテスト。ゲームということに関しては，現状のパソコンではまだまだスペックが低すぎる感じですので，性能競争も悪くはないのかもしれません。少々重いゲームでも60fpsがキープできないというのは，ゲーム機としては完全に落第ですから。その他の用途と比べると，必要なスペックが違いすぎるのが問題ですねぇ。もちろん，ベンチマーク用途の人には，そもそも十分なスペックなんて存在しませんのでまだゲーム用のほうがマシですか。

◆DirectXの情報やMMXを使った画像の加工などの記事が本当に役に立ちました。DirectXを使ったサンプルのミニゲームを何本か入れてほしいです（大きいプログラムだとイマイチ解説が難しいです）。本当にいままでなかった，とても役に立つ雑誌です。オープンソースのプログラムの募集など，ほかの雑誌にない展開をしてほしいです。実力がついたら僕もぜひ投稿したいと思います。これからも内容の濃い役に立つ記事を作り続けてください。

中村剛（19）東京都

いえ，まだまだ十分なことができなくて手探り状態です。プログラム講座はLevel 1のサンプルが大きめになりだしているので，ちょっと注意が必要かもしれませんね。投稿は大歓迎ですのでよろしくお願いします。

◆X68000を初めて手にしてから早くも15年。結局，Oh!Xに追いつくこともなく休刊されてしまいました。そのあいだに結婚，引っ越しがあり，悩んだ挙げ句にOh!Xはすべて捨ててしまいました。しかし，昨年末Oh!X復刊号を手にして正直泣きました。感動と懐かしさと，そして己の無力に。Oh!Xは出続ける限り買います。零式には50万円も用意すればいいですか？そしていつか，この年齢で「いつか」は恥ずかしいですが，いつか開発の一助になりたいと思っています。

八木明（33）神奈川県

期待のかかる九九式と零式ですが，やや開発は遅れ気味かも。値段については見当がつきませんが，一応，40万円くらいということで進めているはずですので，ちょっと上がったとしてもそれくらいには収

まるんじゃないでしょうか。

◆Oh!XはOh!Xなんだなと思う今日この頃です。
井口真秀(23)北海道

そりゃあ，Oh!Xですから。

◆とても面白かったです。値段は高いけど，それ以上の内容でよかった。特に言語処理の話は，こんな処理を使えば数式が扱えるのかと目からウロコ。と，同時にまったく想像できなかった自分が悔しかったです。まだ読み切ってないけど，これからもこういう記事を期待しています。　村上学(24)埼玉県

と，同じことを石上君も思ったわけでしょう。ノウハウの伝承は大事ですね。

◆発売日が延びてやきもきしましたが(4月2日に探し回った)，さすがに内容は充実していますね。PlayStation2。あの記事でいうところの自由度4の壁はとてつもなく厚そうですね。現状の解決方式としては，理論だけは実用レベルまで整えて，あとはマシンパワーのゴリ押しでねじ伏せるしかないでしょう。数学屋さんがまずえらいことになりそう。なにかよい解決方式が見つかるといいのですけど。
中元昌文(23)兵庫県

いずれは越えられる壁だとは思いますが，ノウハウの蓄積が第一でしょうね。非リアルタイムでもできることさえわかればあとはなんとかなるものでしょうけど。

◆広告がほとんどないので高いのはわかるけど，やっぱり値段が高すぎる。また内容も面白くない。X680x0が本当に好きだから復刊1号は買ったけど，Oh!Xを読むたびにX680x0がすでに過去のマシンであることを再認識してしまう。MacやDOS/Vのお絵かきでカラーページを40ページも割くのなら，040や060のアセンブラで白黒ページを40ページ書いたほうがいい。ほかの雑誌でも読めることをOh!Xに期待してないよ。だってほかの雑誌はOh!Xの1/3の値段なんだよ。なんのために3倍もの金を出すんだい？　この厚さでこの値段は同人誌だよ。Oh!Xは復刊したら中身のない同人誌になったといわれないためにも，もう少し企画を考えて編集したほうがいい。X680x0は過去のマシンで話題自体が枯れているかもしれないけど，頑張ってほしい。もし頑張れないなら本当に休刊したほうがいい。　柴田英夫(27)東京都

だからすでに一度休刊してたわけですね。新しいOh!Xはその辺を仕切り直すところから始めているわけで，旧Oh!Xとは少し違うスタンスに立っています。しかし，方向性はまったく同じですよ。X68000やMZ/X1を使って表現していたか，ほかのマシンを使うかだけの違いです。さらにいえば，新旧Oh!Xとも，主眼となる対象機種は「X68000を超えるモノ」です。企画が面白いかどうかはともかく，現状では視野を拡大していくことを主眼に少しずつ目標に向かっているところですから，ちゃんとかたちになるまでもうしばらくお待ちください。

◆「言語処理プログラムを作る」を読んで，SX-BASICを一からアセンブラで書き直したままほったらかしてあったのを思い出しました。一応，インタプリタ部はできていた様子。インターネット端末にしようと思ってiMacを買っちゃったんですが，アップルのサイトにフリーの開発環境が置いてあったのでぼちぼちといじっています。PowerPCのアセンブラもついてるので，なんか遊べるといいんですがね。
石田伯仁(25)神奈川県

SX-BASICはむしろポータブル化を目指しています。どっちかというとSXのタスク間通信がボトルネックでしたし。文字列のやり取りだけというのが無謀だったのかもしれませんが。しかし，フリーのPowerPC用開発環境ですか。それは知りませんでした。なんか面白いことできるかな……。

◆復刊1号が630gで2500円，2号が910gで2800円。グラムあたり4円から3円に値下げですね。
小川博(28)神奈川県

一応，3号目は「安くする」というのを心がけたのですが，なかなか思い切り安くすることは難しいですね。

◆1988年からOh!Xに触れ，休刊となったあいだもCの学習，ミニプログラム，ツールの作成をしてました(すべて未完成……とほほ)。復刊して，この歳になって思わず声を上げて笑いました。なんせうちの私の部屋はX68系本，機器だらけですから。微力ながらもお手伝いしていけたらなと思っています。皆さん値段がどーのこーのといっておられますが，確かに高い。でも金で買えないものも買っているのではないでしょうか？
松本祥司(38)東京都

そういうふうにいっていただけると助かります。高いのは事実ですけど(いくらあがいても安くできそうにないんで，開き直って思い切り贅沢に作ってるのも悪いのですが)。ツールが完成したら投稿して一気に回収してください。原稿料もやや高めに設定しておりますので，掲載されれば万事解決問題なしです。

◆X68000には無料で入手できるGCC，HAS，HLKというすばらしい開発ツールがあった。しかし，Windows上ではCygnus-Win32かdjgppくらいで，おまけにとっつきにくい。そこでLinuxを使ったゲームプログラミングというのはどうだろうか。開発環境はタダだし，VoodooやRIVA系のビデオチップはある程度仕様が公開されているし。かなり読む人間が限定されるような気はするが，こんなことを期待できるのはOh!Xしかないので……。　千葉浩貴(26)岩手県

Linuxがタダというのは正確ではありません。Linuxを動かすための環境を作るのと(ハードディスク買うだけか)，VC++買うのとでどっちがメリットが大きいかなどを考えるとおすすめするのは微妙なところ。ある程度開発のめどが立てばいいのですが，その段階までいくのがちょっと大変ですね。

◆第2号発売おめでとうございます。しかし，そのOh!X春号を見てびっくり。オレのハガキのメッセージ載っとるやんけー。あの頃は学生……若かったもんです。いまでは無事に卒業して単身横浜へ。プロのプログラマとしてのスタートを切りました。入った会社は本当にベンチャー。某MP3エンコーダを開発している会社です。社長はバリバリのアセンブラ使い。PentiumⅢのSSEで4倍速ぅ～などとおっしゃっています。私はといえば得意分野は音声処理，デジタルフィルタでレゾナンス全開ウニウニドンドン～の毎日です。こんな私ですが，いつかOh!Xで記事が書けたらなと漠然と思っていたり……。　花山慎一(20)神奈川県

音声処理いいですね。MMXで4倍速ぅ～な感じで。

◆はい，すみません。復刊第1号のハガキなんですけど，すべて読んでからハガキ書こうと思ってたら卒論とかでこんなに遅くなってしまいました(実はまだすべては読んでなかったりします)。待ちに待ったOh!Xの復刊号ということで私はとてもうれしかったです。X68000の話が少なくなったということで，その分幅広い内容になってマルチメディア時代のいまにとても適したかたちの内容になっていると思います。ゲーム機，CG，C++，Java，DirectXと私のほしい情報ばかりです。2号のほうも広告がないのにあれだけの厚さだなんて驚きです。ま，その分値段もアレなのでアレですが，やはりOh!Xを見てると「まず自分からやる！」ということを教えられますね。私にとってOh!Xは教訓本なのかもしれません。
八東政彦(22)兵庫県

◆第2号おめでとうございます。某社で3Dグラフィック LSIを開発しております。うちの会社はソニーと張りあってるんで，ほかのプロジェクトはソニーさんが出すと「あれを超えろ」とよくいわれてますが，PS2が発表になっても，そういってくれません(せっかくあれを超えるのを考えてたのに)。SIGGRAPH'99にはピクセルフローのボードが出されるそうなので楽しみです。　若山順彦(35)大阪府

3Dの高速化の決め手はバス帯域と並列化ということで，力業勝負になってしまいがちですが，どのようなのを考えていたのでしょうか。Dreamcastに搭載されたPowerVR2がまったく逆の観点「どうやってメモリアクセス量を減らすか」ということに注力しているのと比べると興味深いです。

◆遅ればせながら復刊おめでとうございます。書店で見つけたときのうれしさといったら，もう……。季刊から月刊化へとぜひ期待してます。ガンバレ。
阿知波美紀(26)京都府

月刊化を要望される方は多いのですが，いまのところ現実的ではありません。現状のまま月刊化すると死人が出ます。犠牲者は誰だろうと考えると心当たりはひとりしかいないので，あまりうれしくないです。

◆Tcl/Tkに以前から興味があったので，どんなものか雰囲気がわかってよかったです。ちょこちょこっと

書くだけでそれなりのものができそうですね。Perlはずっと使っているので，Perl/Tkを使えばWindows上のツールをいろいろ作れそうな気がしてきました。
黒武者健一(29)奈良県

◆X68000って2000年問題って大丈夫なんでしょうか？
真田知之(26)北海道

開発者だって馬鹿ではありませんから，1980年代後半に作られたOSでその辺の対応を考えていないわけはないでしょう(Javaじゃあるまいし)。もちろんソフトが独自で年数を省略してたら別ですけど。あとは2080年問題をどうクリアするか……というところまでは，たぶん考えなくてもいいような気がします。

◆なりゆきでNetBSD/X68kのメンテを引き継いでからいつのまにか1年がたってしまいました。最近，NetBSD/Mac68kの書籍が相次いで出版され，古いマシンの再生手段としてのPC UNIXが注目されているようです。残念ながらX68000の開発者は少なく，苦労も多いですが。それにしてもPC UNIXをはじめとするフリーソフト(最近はオープンソースというらしいですが)の関係者に元X68000ユーザーが多いのには驚かされます。おそらくはその大部分の方に影響を与えたと思われる祝氏のご冥福をお祈りします。
箕浦真(28)神奈川県

メンテご苦労様です。フリーソフト関係では，おそらくほかの機種のユーザーが少なすぎるのが目立つ原因ではないでしょうか。50倍から100倍出てるマシンなら，フリーソフト作者の全体数がもっともっと多くても不思議はないのですけど。

◆PS2の記事にはセガ信者の疑惑を持っています。
笹山幸男(32)広島県

なかなか大笑いでした。似たような話がほかでもあって，中森氏などと「どこをどう読んだらそう読めるんでしょうねえ？」と話しておりました。あの時点の分析としては妥当なところだと思います。VRAM容量以外のハードウェアについてはかなり高く評価してありませんか？　その後の非公式な追加情報で「あのデモはちゃんと300MHzで動いてます」とか，意味不明だったデモは「コリジョンのデモです」とか「インタレースVGAしか考えてないでしょう」とか，いくつか明らかになった部分はありますが，とにかくあとは実機を見てからですね。

▲SHIMSOFT　広島県

◆Oh!Xへの期待。オリジナルOSを作ってください。
深川哲光(40)香川県

はい，正しい日本語は「私がオリジナルOSを作ります」ですね。それはともかく，現状の主流マシンであるAT互換機でネイティブなOSを作るのは非常に難しくなっています。もちろん，最新ビデオカードや周辺機器なんかいらない，とかいうのなら話は多少違うのですが。構想だけならいろいろありますけど，開発者が揃っているわけでもないので当分は下準備してるしかないですね。

◆今回初めて読みました。なかなか充実した内容だと思います。普段は，ビジネスアプリばかりを作っているため，なかなか本来のハードの性能を引き出すようなものは書いたことがありません。そろそろDirectXを使ったものを書いてみようかなという感じです。
鈴木芳和(25)岐阜県

そうですね。世の中，プログラマを生業にしている人も多いはずですので，仕事以外のプログラミングというものを楽しむ人がもっと増えてもよいのではないかと思います。ホビー系のプログラムのほうが作ってて楽しいでしょうし。

◆Oh!Xが休刊してしばらくすると子供ができてました。いまはまだ3.5歳ですが「10歳になってもプログラムが組めないようなら山に捨てる」といい聞かせています。3歳にもなると，電源を入れてユーザー名とパスワードを入力し，ログインしてお気に入りのフォルダを開いて踊るくらい当たり前。この様子だと捨てなくてもよさそうです。
下田達也(32)三重県

英才教育ですねぇ。恵まれてるというべきなのかもしれませんが……。まあ10歳なら大丈夫じゃないですか。親子でレンダリングエンジン作るとか将来が楽しみですね。

◆「PlayStation2の予備研究」面白く読ませていただきました。なにができる，できないとあるようですが，結局は使いようじゃないでしょうか。特にゲームなんてものは，どんな手を使ってもよいわけですから。本物を表現するのに本物と同じように計算するというもっとも真っ当なやり方がそのままできるのは確かに凄いことだとは思いますが。
花井章能(28)東京都

まあ，最終的に辻褄があっていればよしというのがゲームの基本だったわけですが，その枠を超えていかれるということみたいなので，どんなもんだろうと注目が集まっているわけですね。ある意味では，まさに夢の実現ですし。

◆パーソナルコンピューティングという目的でOh!Xを作っているのなら，コンシューマ機の内部情報は不要なのではないか。
田添政勝(29)兵庫県

コンシューマ機の環境は閉じた世界ですが，本来はパソコンとリニアにつながっているべきものだと思います。現状はともかく，できるだけ接点を広げる方向で展開を考えています。また，表現物としてのコンシューマゲームはもっと多彩な側面から評価されてよいものだと思いますし，その機構を知っておくことはあながち意味のないことはないでしょう。さらにいえば，それを作る側に回る人も少なくないのが実情ですし。

◆中学生時代に憧れていたX68000を最近秋葉原の某中古ショップでゲットしました。一昨年も友人から1台もらったのですが，電源が壊れており悲しい思いをし，やっと手に入れたのです。当時は友人宅で「ゲームマシン」となっていましたが，あの当時でアーケードを忠実に再現していたマシンはX68000がいちばんだと思いました(アフターバーナーの煙は単色でしたが……)。なかでもスペースハリアーは感動の嵐！「HAYA-OH」はアーケード版にはいないボスでしたが，HAYA-OHを倒すために，いや，再度HAYA-OHを見るためにX68000を購入したようなものです(サターン版にはいなかったし)。いまではAT互換機を仕事用に使っており，パソコンに感動することがあまりないのですが，X68000のマンハッタンシェイプのような魅力的なマシンが今後出てくることを期待してます。そういう意味で零式には大変期待してます。
林純一郎(26)岐阜県

連射装置全開で戦うスペースハリアーはアーケード版より爽快でした。初めてHAYA-OHを倒したときは感動的でしたね。最初は見とれてるうちにやられてましたし。

◆「ネットワークゲーム」の文字につられて買いました。いま作ってみたいなと思っているので。こうやっていろいろな方向からの記事があるというのは見たことありませんでした。凄いですね。旧Oh!Xは知りませんでしたが，読者レベル高いんですね。
折野裕一郎(20)東京都

ネットワークゲームは基本的な部分だけでちょっと消化しきれずに終わった感じですので，そのうちリベンジしたいところです。

◆Oh!X復刊2号も無事発売とあいなり，めでたい限りです。作り手が命削って作ってる本はやはり禁断の味がするようで，読み手のほうも睡眠時間を削ってフラフラになりながら読んでいる次第です。今春は愛読していた廃人御用達雑誌が会社ごとなくなるという憂き目にあい，いまや貴重なるマニア系雑誌にはしがみついていく所存ですので，今後ともよろしく。
鈴木文章(30)茨城県

削りたくて削ってるわけではないんですがねぇ……。

◆おお，アスキーにも負けない厚さにびっくり。広告の少なさに二度びっくり。ぎっしりと記事が……。ちょっと正直読むのがしんどいか。さあさあなにが生まれてくるのか，育つのか，どこへ行くのかという雰囲気が懐かしくもあり，うれしくもあり……。祝一平氏のもう読めないと思っていた記事が載っていたことに驚きつつ，ご冥福をお祈りします。

波多野雅章(37)大阪府

◆Oh!Xがまた動き出したからというわけではないのですが，流れ流れて有田先生と同じ建物に学生としてきました。それにしても，モニターや広告を出しているスポンサーの苦言をここまで書くOh!Xってやっぱりすごいパワーです。 佐川正人(30)愛知県

いえ，単になんも考えてないだけのような……。広告は最初から「入らないモノ」として試算して作ってますので問題ありません。もっと広告が増えると安くもできるのは事実ですけど。

◆仕事のあいまに本屋に買いに行きました。見当たらないので本屋のおじさんに聞いたら「毎月18日発売だったよねぇ。まだきてないよ」といわれました。そのあと説明したのはいうまでもありません。

田村和広(27)東京都

◆DOS/V magazineに「Oh!X発売日4月2日」と書かれていたので4/2～4/5にかけて本屋に入り浸りましたが，本を発見できず。「もう完売かな～」と悲しく思っていたところ，4/17に発見。しかも「本日発売」がついていました。とりあえず本を買えてとても満足です。部署が移って10年前に(自分が)X68000で作ったプログラムを，再び見ることになりました。本の復活にあわせたようで，なんかむずがゆい気がします。

杉本浩(33)北海道

◆現在自作のネットワークゲームに挑戦中でしたので，ナイスタイミングの特集でした。国産モノはどうもネット系が弱いので，新風を呼ぶ国産モノを目指して頑張ろうと思います。β版ができたら送りますので気長に待っていてください(いつになるかわかりませんが)。心残りは祝さんに見ていただきコク評をいただけなかったことです。祝さんの遺言級ゲームの話，いまでもアレを超えるコラムにあったことありません。心よりご冥福をお祈りいたします。

佐竹一生(33)福岡県

◆Oh!X復刊おめでとうございます。Somethingの「X」として，さまざまな話題を取り上げていってほしいと思います。価格については広告も少ないところから，これくらいはしかたないと思います。読み捨てる雑誌が多いなか(私はパソコン関連の雑誌はほとんど捨てませんが)，読みごたえのある永久保存版MOOKとしてのいまの位置づけをキープしてほしいと思います。近い将来自分も参加できたらいいなと思っています。

寺沢昌記(32)東京都

◆忘れた頃にやってくる雑誌Oh!X。店頭で見つけてついつい買ってしまいました。はっきりいって難しいですが，面白いです。零式の記事は意味のわからない単語も多く出ますが，その中にも未来を感じさせるパソコン像が伝わってきます。月刊化希望。でも価格が高いのがきついなあ～。 倉本恭裕(27)北海道

◆他誌，専門誌などにならないような特殊な記事を期待します。 岡谷憲光(27)岡山県

特殊って……。

◆「特別企画　もう一度CPUについて考えてみよう」は特に面白かったです。CPUやそのアーキテクチャに関しては詳しいつもりでしたが，知らなかったことが結構ありました。「インターネットセキュリティ事件簿」も面白かったです。事例をもとにしているので，わかりやすかったです。フェスタ68に行ってきました。込んでいて暑くて疲れたけど元気もらってきた感じです。

武藤一文(26)静岡県

え，静岡から行ったのですか。それはお疲れさまでした。

◆丹氏の記事がなかったのが残念でしたが，超一流ソフトであるグランツーリスモに関する制作者本人の声が聞けてよかったです。これからも素晴らしいソフトを作っていってください。個人的にはモータートゥーングランプリ2のVanityが気に入っているので，バイクも登場させてくれるとうれしいです。実際にほかの分野で働いている人の声という感じで(DoGAのテレビアニメの作り方など)いいです。物理学を専門にしているという三沢氏がレースゲームなどでの物体同士の衝突の計算は大変というのなら相当難しそうですね。それを感じさせない世にあふれるゲームはよっぽど凄いんですね。 福田強(25)神奈川県

「ちゃんとやると大変」という感じですので，それなりに済ますならなんとかなってるということでしょうか。

◆わ！　厚い！　出るたびに厚さが増していくのが新生Oh!Xなのか～？　LIVE in'99がなかったのがちょっと残念でした。 松本健一(26)静岡県

今回はあまり厚くしない予定でいたのですが……。薄い本のつもりで先に定価決めちゃってたんで，膨れ上がってしまってちょっとヤバめな今号です。

◆いまさらですが復刊おめでとうございます。Oh!Xが休刊したことを知ったのはつい最近のことで，ああこんなことなら最後までつきあえばよかったなあと後悔していたところ，さらに5年間ほったらかしにしていた愛機EXPERTが故障していたことに気づきました(電源の故障)。修理したくてもお金がなく，なかばあきらめてました。そんなとき，本屋のパソコン関連コーナーの前を通ったら目の前になにか見覚えのあるロゴが映ったような気が。振り返ると，そこには「Oh!X復刊」の文字が！　うちのX68000が動かないことも忘れて購入しました。そしてこの復刊第2号にうれしい記事が！　ATX電源接続キットを使えばうちのX68000もよみがえる！　でもATX規格ってなに？

江ヶ崎貞行(25)千葉県

とりあえず，修理見積もりを取ったほうがいいような気がします。ATXは最近のIBM PC/AT互換機で使用されている電源ユニットです。パソコンパーツショップで買えますが，普通1万円以上，安いのを探せば3000円くらいですか。どうしてもある程度の出費は必要です。

◆アクションゲームなどが楽しいのはわかります。CG性能が上がり，どんどん面白いものができてくることでしょう。そういったものについて，現Oh!Xは十分対応していってると思います。一方，プレイヤーを感動させたり，入り込ませたりするのにシナリオや物語の構成力は必要でしょう。これについていまのOh!Xでは対応できてませんよね。僕は物書き(小説，脚本)を目指しているので，ゲームにおけるシナリオの未熟さが身に染みてわかります。コンピュータからは少し離れるかもしれませんが，そういった記事を読みやすい文体で初歩から掲載してはどうでしょうか。僕も書きたいのですが，実績，実力ともに……。ぜひゲームのシナリオライターではなく，映画のライターに書いていただきたいです。 Ikep(22)東京都

とりあえず，今回のストーリーのでっち上げ方とかはどうでしょう。答えがあるものではないので，書ける人はほとんどいないような気がしますけど。

◆市川氏の記事が好きです。ピンボールゲームといえば，セガの「ロビンフッド」や「ボイジャー」を取り上げてやってください。セガのピンボールゲームは「マッチがない」「クレジットの上限が15まで」とミソのつくところが多く，昔から社風が変わりませんし，当時小学生であった私の仲間から敬遠されがちだったとは思いますが，ゲームアイデアは斬新で面白かったです。あと，IC基板になる前のピンボール機は内部に回転板(1軸数枚)があり，回転してスイッチが切り替わるたびに青白い火花が飛んで美しかったです。機械フェチとしてたまりません(完全稼働台がないものか)。

佐藤博是(32)長崎県

このコーナーでは皆さまからのご意見を募集しております。綴じ込みのアンケートはがきや，e-mailなどでご意見をお寄せください。
e-mailでの宛先は，
aueki@softbank.co.jp
となります。
また，投稿イラストも随時募集しておりますので，
〒103-8501
東京都中央区日本橋箱崎町24-1
ソフトバンクパブリッシング(株)
DOS/V magzine編集部内Oh!X宛
でお送りください。皆さまの参加をお待ちしています。

編集室から from E・D・I・T・O・R

SHIFT BREAK

▶「オルタナ・ピコピコC言語講座」を担当した飯田伸一です。Oh!Xが復刊し，原稿を書かせてもらうことになるなんて，再びパソコンを触り始めた1年前には，想像もできませんでしたよ。人生って，なにが起こるのかわからないものですね……。
（飯田）

▶今回でOh!X復刊3号目ですね。Oh!X って昔の雰囲気を残したまま新しい時代に適応したかたちで復刊したわけですが，順調に形となっていっているのが実感できます。こういったスタッフやライター，読者のみなさんが一体になって熱気が感じれる楽しい雑誌は最近少なくなってきたのでみんなで絶やさないように頑張っていきたいですね。さて今回の号で，私が描かせていただいたCGのような光景は，誰もがよく体験するようなことだと思います。毎日が単調になり少し惰性で生活をしているときに新鮮な刺激が入るというのはちょっとよいものです。私自身も初めてX68000で制作したCGは，地球や火星の周りを，飛行機が飛んでいるといった，ありがちで，単純な作品でした。でも，よい思い出となり，いまでは私の大切な作品のひとつになっています。最後に，前回のOh!Xに私の記事が掲載されたということで，友人や知人からたくさんのお祝いをしていただきました。この場をお借りしてお礼を申し上げたいと思います。本当にありがとうございました。
（田中）

▶個人の力でマンガを描くのは大変な重労働です。今回の原稿が本気でマンガを描こうとする人のなんらかのお役に立てば幸いです。（森川）

▶もう1999年だし，7月過ぎてるしなんだけど（8月だとか9月って説もあるけど）世界はとっても平和。降ってくる気配すらなし。ま，滅びるとはひと言もいってないし。私はここんとこ激務が続いてて，しょっちゅう目が回る状況。いっそ大王が降ってくれて，みーんなブチ壊してくれると仕事が楽になるなぁ……ってそんなコト考えちゃいけませんね。海より深～く反省。（霧）

▶Oh!X夏号の原稿が終わったすぐあと，結婚式出席のために新潟へいきました。ビジネスホテルに泊まるのもつまらないので，駅からバスで40分ほどの温泉宿に泊まることに。綺麗な温泉宿だけど平日なので宿泊客はまばら。大浴場は独り占め。温泉のお陰で原稿描きでボロボロになったお肌は温泉でツルツルに。お酒はおいしいし，食事もおいしいしで，のんびりしまくりました。ああ，また旅行したい。
（野沢絵美）

▶日本ではやっと今年最初のピンボールの新作が発売される。Ballyブランドの第1弾！ Revenge From Marsだ。ピンボールにもついにマザーシステムのアイデアが持ち込まれ，フィールドとソフトを変えると新しいピンボールになるというもの。ただ，1つひとつの台がまったく違うのがピンボールの長所でもあるだけに今後どうなるのか大変興味深い。次回はPinball2000の紹介とフリッパーの構造について記す予定。http://pinball2000.com/
（市）

▶描いている最中に，ATXマシンを組み換えたり，BIOSアップデートしたり，そのアップデートに大失敗したり，修復してもらったり，もう大変で楽しい日々を送っております。というわけで，4コマも完成度が高くなくて（でもいつもどおり）申しわけありませんでした。仲間内に約束した新キャラも描けなかったし，まだまだです。とりあえずこれで，自分のホームページの更新にようやく取り組めるぞー。（岡村）

▶もうだいぶ前の話になるが，家庭内LANを組んでみた。TAが普通のTAなので，Linux（Slackware 3.5）をルーター代わりにして怪調？に動いている。これを機にUNIXのプログラムをぽちぽちと組み出してみて，初めてUNIXの簡潔さと強力さに気がついた。FreeBSDなんて10行もあればCで常駐ものが書けてしまう。DOSじゃ考えられん。LANカードもハブも安くなったし，UNIXも気軽に扱えるようになったし，いい時代になったもんだ。しかし，これらをどう生かすかはユーザー次第なのである。
（UNIXはアマグラマー Moz）

▶家が信州にあるんで夏でもクーラー/エアコンなしでコンピューターの前に向かっていたりします。扇風機も一応あるけど滅多に使わない。しかし，さすがに8台以上使用していると暑い～。Mac OS X Serverなんぞ入れたもんだから常に稼働中のマシンが4台。目の前にあるのは11台。最終的には16台かなあ……。毎回買ってくる市役所専用の納品書の穴が上にずれているのが気になって親が塩尻市役所へ。市役所からの回答。「いままで気づきませんでした」どう見ても20年以上は使っている納品書のハズなんだけど気づかないのかあ……（誰もいままでいわなかったようだし）。市役所不思議也。（古籏）

▶時流に乗って（?）Linuxマシンを導入しました。おうちでUNIXは嬉しいですね～。やっぱりUNIX系のOSが見通しが利いて好きです。ご多分に漏れずネットワーク関連でいろいろ引っかかってますが，文句を垂れつつ結構楽しんで環境整備してます。
（うぃ）

▶8月7日に岡山の小学校にて「夏休みCGアニメ教室」を開催しました。この経験を生かして，「小学生のためのCGアニメ入門」という本を執筆する計画。……だれが買うの？ 自費出版か？ 最近ネット声優（自宅で録音し，インターネットで声のデータを送る）の方々と交流しています。ネット声優は活動の場を求めており，CGアニメ作家は声優不足に嘆いています。そこでWeb上に，この両者をつなぐ声優紹介のページを作ろうと計画中。年内に稼働して，12thCGAコンテストの応募者は声優に困らないようにしたいものです。
（かまた）

▶発売日が大幅に遅れ，締め切りもずるずると延び，こんなことならIMやる時間も十分あったなぁ，と。まあいいか。全然関係ないが，ちょっとした気の迷いで車を買った。都内で車なんて，ランニングコストが高い，移動の時間が読めない，買うだけ無駄というのが（U）氏と共通した私の見解だったのだが。PenⅢマシンも買ったりとここのところ出費が激しいのだが，なんとなくe-oneが気になる。ちょっとiMacと一緒に並べてみたくなったりもする。って，iMacも持ってないんだけどね。なんとなく経済観念が狂ってきているぞ。世間は不況だっつうのに。（I.K）

▶Oh!X1999春号の西尾ゆきさんの記事で「バニーの制服のせいでキャバクラ状態」と書かれていたので「バニーさんいるのはキャバクラとちゃうやろー。キャバクラにいるのは飾りを取ったセーラームーンみたいな服のおねーさんやろー（そうか？(^^;)」とメールで突っ込んでみた。するってーと「じゃあ，バニーさんがいるのってなんなの？」と突っ込み返された。さて……，ロイヤルとかエスカイヤ・クラブってのはなんになるんだろう。少なくとも，キャバレーではないし（ねーちゃん馴れ馴れしくないし），クラブ……じゃないよなぁ。もちろん，若者が集うクラブ（イントネーションが違うよ）でもないし。やっぱり「バニサロ」かしらん。……あのね，間違っても「ふーぞく」じゃないんだからね。そりゃあね，バニー姿でお出迎えってお風呂屋さんあるらしいけど……聞いた話。……札幌らしいけど……聞いた話。ホントに聞いた話ですってば。あ，あのね，だからね，「風俗に詳しい（で）様」とか，そーゆー恐ろしいフレーズつけるのはやめてね。嫁がメール読んだら殺されますって。まじで。って，ああ，どうして今月はこんなことばかり書いているんでしょー，私。え，減殺ですか？
（やっぱり銀座リッカー会館でぼくと握手！（で））

▶どーも。善です。復活版Oh!X第1号にて紹介したチームプレイ主体の3D対戦シューティング「TANARUS」ですが，やはり運営維持できなくなり閉鎖されてしまいました。やっぱり課金制の有料対戦ゲームサイトは日本では難しいようです。……と思ったものの，なんと同じソニー系列のSo-netが運営を引き継ぎ復活を果たしたようです。意地ですな。ということで，8月一杯はテストを兼ねて無料開放されているので参加してみましょう。URLはhttp://www.so-net.ne.jp/partycrew/です。「ポピュラス・ザ・ビギニング」，20世紀最後の最高傑作ゲームと呼ぶにふさわしいゲームでした。とにかく買ってやらなきゃだめです。PS版のデキはかなりヤバイのでPC版をプレイすることをおすすめします。Pentium II/300MHz，メモリ64MB，RIVA128程度のスペックがあればちゃんと動きます。マウスはお絵描きができるくらいのちゃんとしたレスポンスのものを用意しましょう。うまくなったら対戦サイトで会いましょうね。現在は，私はポピュラスの開発元のBullfrogの最新作「ダンジョンキーパー2」を攻略中です。これまた対戦が熱いゲームですよ。さてさて，今回，私にしては珍しく，デジカメ関係の話題に首を突っ込んでいます。でも，私だって「デジカメの西川さん?」と人違いされるくらいですから，デジカメと無縁ではいられないんですよ。わっはっは。長らく使ってきたリコーのDC2-Lは3回目の修理でとうとう修理費見積もりで2万円と算定されたんで，もう使うのやめました。工賃1万円はいくらなんでもひどいですぜ，リコーさん。で，ソニーのサイバーショットプロに買い換えました。世の中の主流は200万画素オーバーですが，150万画素のコイツで今のところ満足してます。

(西川善司，http://www.z-z-z.gr.jp/zenji/)

▶独立してからというもの，ずっと着物派になってしまった。最初は別に深い理由があったわけでもないのだけど，着始めると，なるほどよくできていると感心する。冬場は暖かいので愛用していて，夏になったらどうしようかと思いながら，そのままずるずるときて，結局浴衣1枚でいたり，薄手の着物を着て出歩いたりするのだが，結構暑くても耐えられるということに驚かされる。考えてみるとだいたい暑い国の服装というものはなるべく肌を露出せずにゆったりとさせたものが多い。洋装の場合は基本系は「寒ければ覆う」「暑ければ肌を出す」という方向なのだけど，それは高緯度地方の常識なのだろう。直射日光は遮って，肌の周りには空気の断熱層を作り，動くたびにそこの空気が移動するという仕掛け。ついでに木綿なら汗を吸うし，気化熱で冷却されるという仕組みと考えれば，なるほど理にかなっている。冬はというと，襦袢/着物/羽織と重ね着。しかも着物や羽織は袷になっているので，風を通さない断熱層をたくさん抱え込むわけで，暖かい道理である。ところで，先日とある方の随筆集を買ってきた。昭和の10年頃の映画館やら劇場にすでに冷房が入っていたというのは初耳であったし，20年頃にはすでに着物姿が激減していたようだ。この頃から無粋の道を一直線に突き進んでしまったのだろうか。

(Kuw)

▶DCにしか採用されてなく，長い間神秘のベールに隠されていたSH4が，ついにWindowsCEマシンに搭載され，我々の前に姿を現した。注目の性能はほぼ公表値どおりの値が出ているようだ(僕は200MHzで360MIPSというのを全然信じてなかった)。しかし，ユーザーモードでSH3とオブジェクト互換性がないのはいただけない。具体的にはPC相対アドレッシングの動作が異なるようであるが，噂では40回近いマスク改版をしたというSH4なら修正する機会はいくらでもあったはずなのに。謎である。なにかと話題の多いPS2であるが，9月のTGSでいよいよお披露目だそうである。T社から聞こえてくる噂ではPS2のCPUであるEmotionEngineの歩留まりは相当ひどいもので，年内の発売は絶望的とされていたのだが，久多良木社長の神通力は物理法則さえも覆すことができるのか。でもPS2には期待してます。SCEファンの熱狂的支持を集めているPS2とは対照的に，任天堂ファンはDolphinにはまったく期待していないように見える。IBMが早々と拡散途中のCPU(Gekko)を取り出してチップ写真を公開したにもかかわらず，64DDなどの過去の苦い思い出が慎重な態度を取らせているのだろう。周囲の状況を考えると，僕自身は予定どおりに発売されると思っているのだが。ところで，システムが「ドルフィン」でCPUが「月光」というからには，グラフィックチップは「海パン」になるのだろうか。

(幸恵ちゃんに会いたいA.N)

▶休日に近所のバザーに行ったら懐かしき電子ブロック発見。信じられない気持ちで10円玉2枚を置き，早速，嘘発見器を作ったら「嘘」の音が鳴った。感激。 (有)

▶組織上の問題で春から急にDOS/V magazine本誌に編入されることになり，年度当初の予定がすべて狂ってしまった。いまごろはOh!Xは終わって，ビデオカードの本を出して，K7本にかかろうかというところのはずだったのだが……。それはそれでハードなんだが，なんか話が違う。少なくとももうひとり人は増えるはずだったので，それに応じた予算計画を立てたのだが，人はいなくなっても予算だけ残っている。私自身は，よく見ると予算上はDOS/Vムックチームのままということで，さらになにか変な感じ。おかげで/Vmag.の共有フォルダにも入れなくなってしまった(以前は入れてたのだが)。とにかく/Vmag.本誌もやるが，Oh!Xをやるときは抜けるということで了承してもらっているのだが，しかしこの分でいくと年間8カ月くらいは本誌のほうに手が回りそうにない。いいのか? (U)

無理は承知……なんだけど

立体表示とデジカメ関係。1999春号に入りきらなかった企画を2つ，ようやく掲載することができました。でもそうかと思っていたら，またまた今号からあふれてしまう記事が続出しています。当初の予定よりも48ページ追加したのですが，「あれれ?」という間にどんどん増殖して，入れなきゃいけないものまで削らざるをえない状況になってしまいました。今回落ちたものは次回回しということで，もう少し練り込んでお届けします。

内部的な事情となりますが，不定期刊とはいえ，DOS/V magazine本誌との絡みがあるので，実質フリーになるのは1カ月程度。その間も/Vmag関連でのインタラプトが入るので……。ひょっとして素直に隔月刊くらいのほうが楽なのではないかと思い始めている今日この頃。

物量が私の管理能力を超えてきているのが問題でしょうか。弱音を吐くつもりはないのですが，どうしても行き届かない点が増えてきていることについてはお詫びしておきたいところです。残念ながら，現状ではもうちょっとちゃんと仕上げるだけの余力がない状況ですので。

さて，制作側の事情はともかく，データ的なところも少し紹介しておきましょう。Oh!X復刊号と第2号の売上データを前にしてみんな首をひねっていました。復刊号と第2号の発売後の店頭売り上げ数のグラフがぴったり同じカーブを描いて重なっているのです。線はどう見ても1本しかありません。なにかの間違いではないかと数値を見ると，ちゃんと微妙に違うのでデータの間違いというわけではないのです。もちろん日によって売り上げにバラつきがあって当然です。定期刊行物ではないのですし，春号の発売日は遅れてしまったのですし，なにより値段もかなり違います。それでもなお，綺麗に前回の推移グラフの上をトレースしているデータが出てきたわけです。

いくつか可能性が考えられます。

・値段は高くても安くても変わらない
・本来流動的要素であるはずの新規読者がほとんど存在しないか，割合は一定している
・発売されてから，それに気づいて買うまでの時間がほとんど同じ

だいたい固定層があるというのはわかるのですが，2番目の項目はいただけません。今回は薄くして値段を下げてみたらどう変わるかというデータがほしかったところなのですが……。厚さは前回とほとんど変わらずということで，なかなかうまくいかないものです。

microOdyssey

時代は少女小説である，と思う。少女マンガはある程度市民権を得ているようだが，少女小説というとまだギョーカイでも肩身が狭い。「え〜っこばるとぉ？」などと驚かれることがある。この反応も8割くらいは正しい。大半は読み飛ばして捨ててもいいんだが残りの2割が問題なのだ。小説不毛の時代でもこのジャンルはやけに元気がいい。毎月毎月どんどん新作が出てきて，新人作家もたくさん登場してきている。これらはまさに玉石混交。ファンタジー系のライト小説も元気がいいが，そっちがたいていバカなだけで面白くないのに対し，少女小説系で当たりを見つけるとなかなかに深いモノがある。違いは「型」があるかどうかだ。ごった煮のなんでもあり状態だからこそ，生まれてくるモノもあるのだ。不安定だったり，勢いだけだったりすることもあるが，勢いがないよりはいい。

基本的に，少女マンガの「深い」作品は，そこらの文学作品よりも深遠な世界を描ききってみせる。絵と台詞というメディアの表現力を駆使して展開されるそれに比べると，文学というのは無力なメディアにも思えるほどだ。多くの才能が投入され，世界に誇るべき日本の文化を築き上げているといっていいだろう。クオリティの高さは別に「少女」マンガに限ったことではないのだが，よりテーマに制限がなく，ときおり妙に突出した作品が出ているのは広く知られていることだろう。

基本的に「マンガ」であるということのほうが「文学」であるより表現の幅が広いと思われていたこともあり，マンガのほうがメディアとして有利な位置にあった時期が続いていたように思う。「マンガのほうが自由度が高い」と思い込んでしまっている人がいても不思議はないかもしれない。一時は「制限は皆無」といわれた少女マンガだが，レディースコミックが登場するに至って，多くのどうしようもない作品に交じって，少女マンガの枠内では絶対に描けなかったであろう傑作がいくつか生まれている。そこで初めて少女マンガの枠が認識される。確立された分野というものは，どうしてもそれ自体で枠を持ってしまうものなのかもしれない。そのなかで文化は成熟し，閉塞する。そういう意味で日本文学の閉塞感というのなかなかに強固だったと思う。

が，事情はだんだん変わってきているのではないかと思う。マンガも小説も，だいたい同じ人種がやっていることで，方法論が違えども内容自体はさほど差がつくはずのモノではないはずなのだ。現代の日本人など，アニメで育ち，マンガや小説はもちろん映画，ゲームなどあらゆるメディアの洗礼を受けて鍛えられている。「文学」という硬い響きが文学自体の可能性を閉ざしていたのではないかとも思われるのだが，若い世代にはそのようなものは関係ない。ファンタジー系の流行や非常に都合のよい世界設定をしても許される環境，なにをやってもいいようなざっくばらんな状況ができてくると小説本来の力が発揮されてくる。これは一時SFということで，許される範囲が広がると文学的にも興味深い作品群が続出していたのにも似ている。日本におけるSFはその自由度自体と文学たらんとしたゆえに自滅していったメディアではないかと思うのだが，いまは世代が変わって，与えられた自由度に臆さない作家群が新しい次元を切り開いているように思われる。それが現状の少女小説の世界だ。

人材はほぼ同じなので，基本的に少女マンガと同等の深みに踏み込むのはなんでもないことだ。ビジュアルによる表現を持たない分，ときに，絵では描けないところまで追い込んだ表現をしていく。同じ展開をするあいだにビジュアルな情報量ではマンガが有利だが，ロジカルな情報量では文学が有利だ。端的にいえば，やおいマンガとやおい小説の表現世界の違いを意識的に使える作家なりが存在するということだ。絵として表現しにくい感性の部分を武器として使え，許されたダイナミックレンジをフルに使って，ときに残酷なくらいに人間の心理を切り分ける。文学者が様式や伝統手法という高級言語で記述しているのに対し，彼女たちはフルアセンブラでガリガリに書いているといってもいいかもしれない。しかも確立されたアニメライブラリやゲームライブラリを自由にインクルードできるのだ。

必ずしも全体の完成度が高いとはいわない。が，ポテンシャルはきわめて高いのだ。全体にムラが多くて安定しない傾向が強いのだが，そういった勢いのなかで突出した作品が世に出てくるのも時間の問題であろうと思っている。

それぞれのメディアはそれぞれの表現のかたちというか，定石をノウハウとして蓄積していくことでより高い次元に到達しようとしている。半面，手法が確立されていくにしたがって，可能性自体も制限されてくることがありうる。枠組みは失敗しないためのレシピでもあり，それを超えることはある意味，異端である。原理的に，新奇さだけを狙ってもほぼ確実に失敗は見えている。

行き詰まれば根底にある前提を変えるしかない。同じ土台の上での成功則は見切られているからだ。というわけで，まず，新しいものを生み出すには新しいものを受け入れる土壌を作るのがなにより大事ではないかと思うわけだ。（U）

次回予告

Oh!X 1999年秋(冬？)号 11月吉日発売予定

特集　未定(携帯系プログラミング？)

特別企画　とりあえず，オリジナルATマシンを組む

特集2　ゲーム制作系

Oh!X 2000年冬(春？)号 2000年秋発売予定

特集　未定

Oh!X 1999 夏号
DOS/V magazine編集部　Oh!X制作実行委員会編

■1999年9月20日発行　定価2300円(2190円+税)
■発行人　岡崎眞
■編集人　稲葉俊郎
■発行所　ソフトバンクパブリッシング株式会社
〒103-8501　東京都中央区日本橋箱崎町24-1
編集部　☎03-5642-8189　FAX 03-5642-3429
販売局　☎03-5642-8100
■デザイン　クニメディア(株)
■印刷　クニメディア(株)

乱丁，落丁，CD-ROMの破損はお取り替えいたします。小社販売局までご連絡ください。
雑誌　65814-82

Oh!X 1999 夏号

愛読者カード

郵便はがき

料金受取人払

日本橋局承認
3181

差出有効期間
平成12年5月
14日まで

1 0 3-8 7 3 0

346

(受取人)
日本橋郵便局
私書箱358号

ソフトバンクパブリッシング(株)
DOS/V magazine 編集部 内
Oh!X 1999 夏号 係行

フリガナ	年齢	性別
お名前	歳	男・女
ご住所 〒		
e-mailアドレス： TEL URL：		
勤務先名 (学校名) 使用機種/OS	パソコン使用歴 年	
購読パソコン誌	プレゼント希望番号	

1999夏号通巻170号

満開の電子ちゃん

作・え　岡村祭

どお？この肌の色
スケルトンでしょやっぱ時代は
俺なんかDVD内蔵でこのスレンダーボディー

・・・・なげかわしいね
まったくだ男は中味！質実剛健よ
その通り

キャー
スタイリッシュ
かわいーー
インテリアとしてもイケてない？！

すてきすてき家に来てー
ずるーいあたしが先よ
キャキャ
・・・・・・
・・・・・・
・・・・

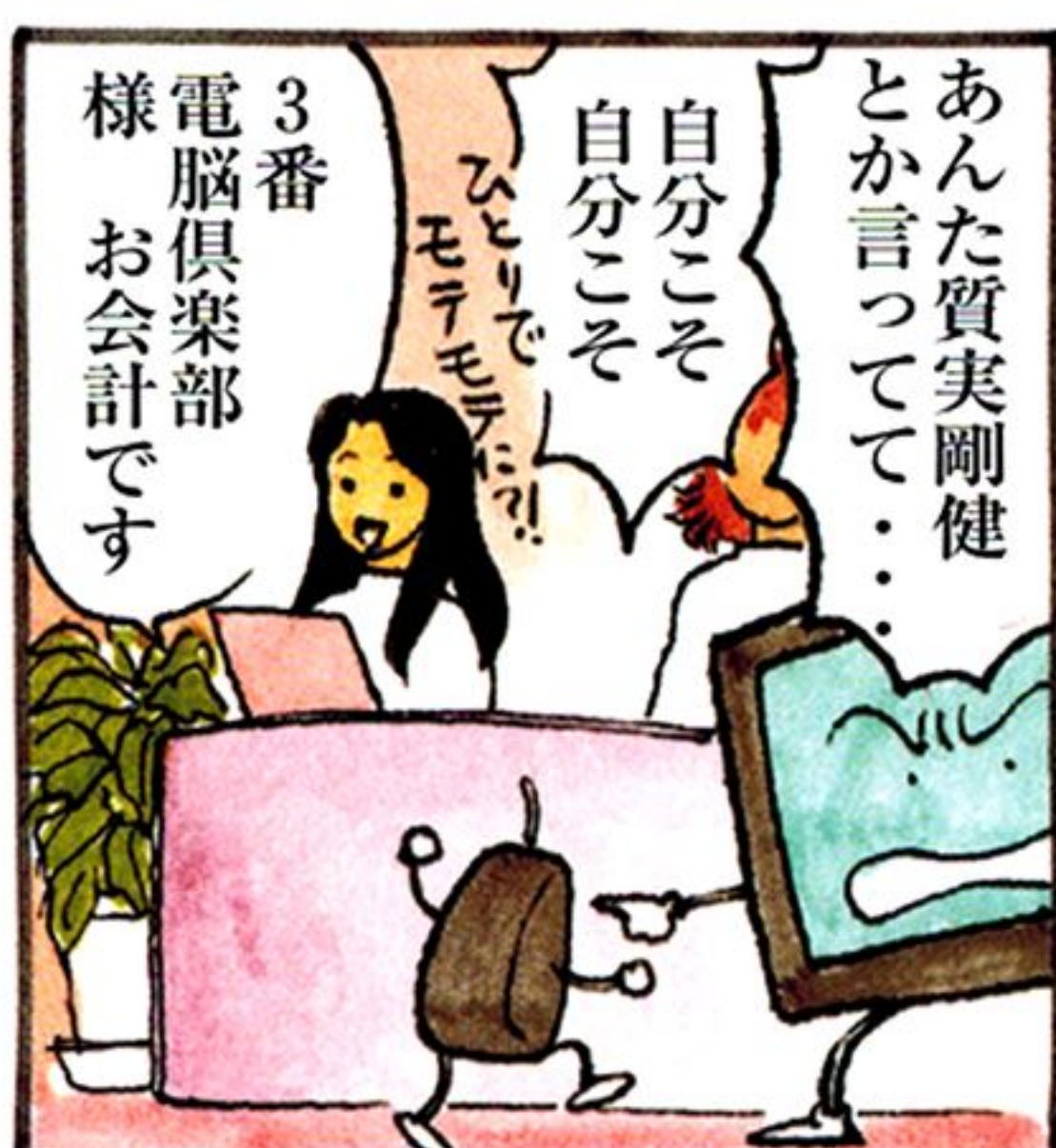
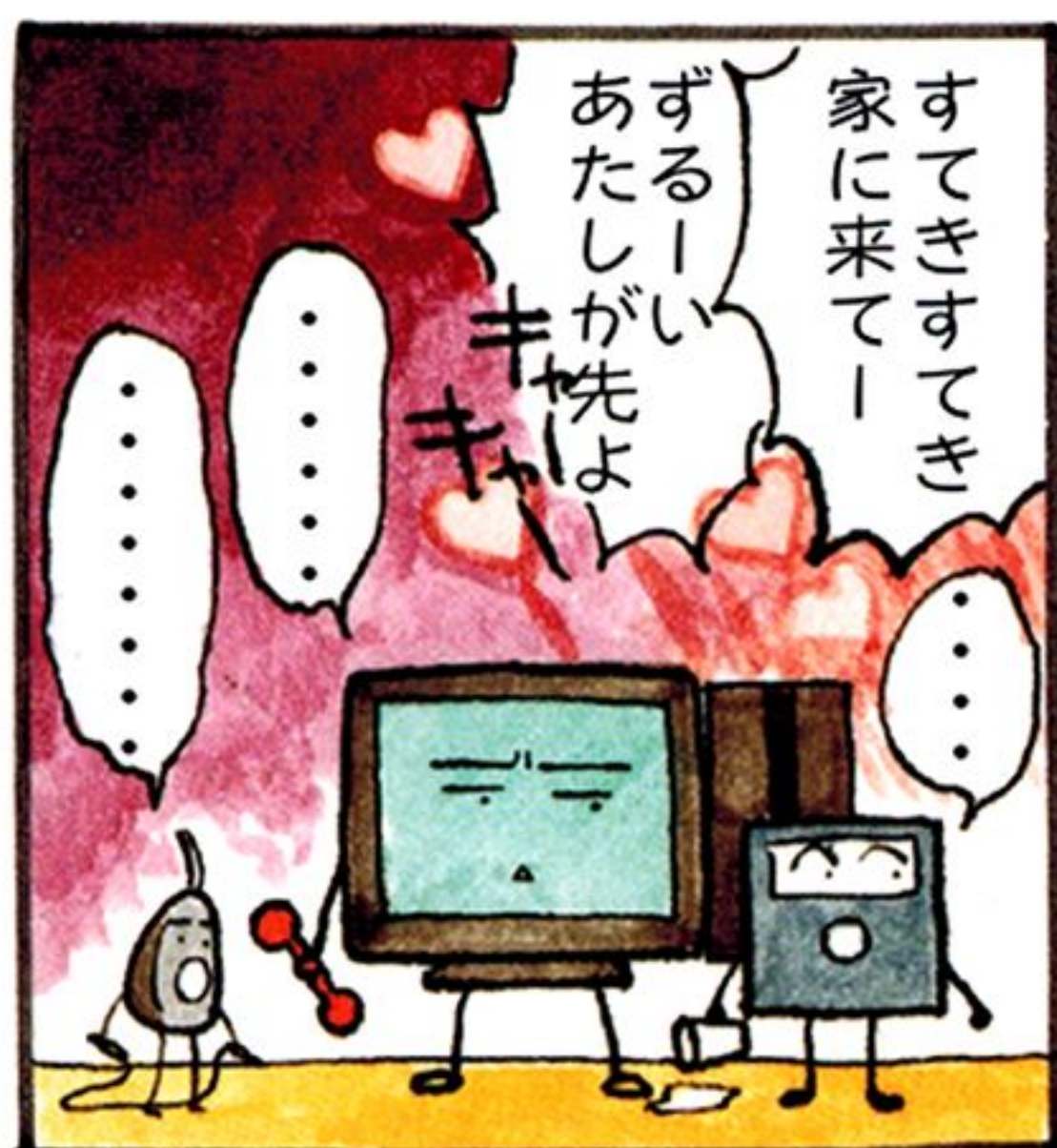
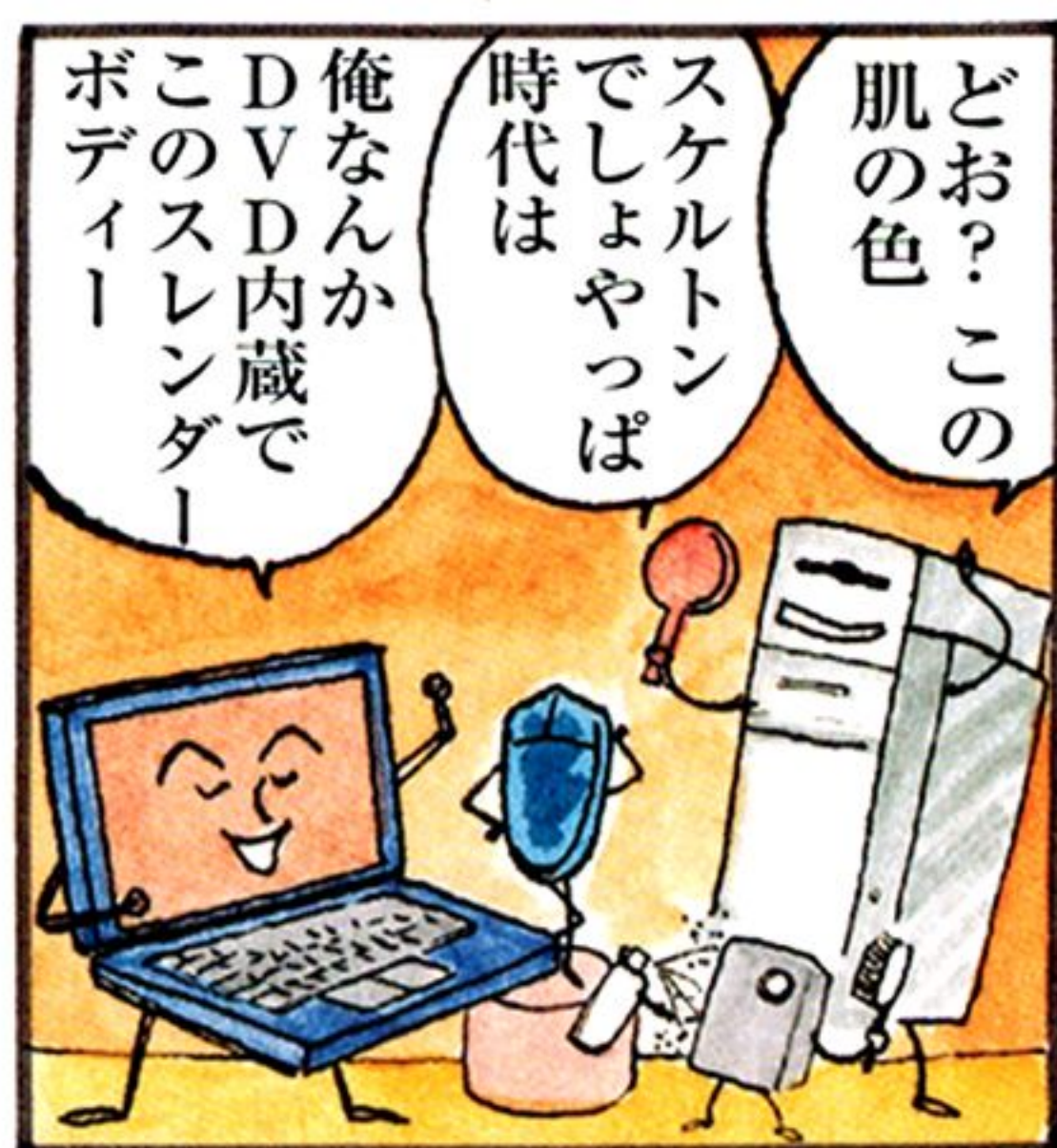
6818番マウス様お待たせしました
脱毛
美顔
メンズエステ
夏のボディエステでみ
入会受付中

こちらへどうぞ
あっあーっ
あっやばっ

あんた質実剛健とか言ってて・・・
自分こそ自分こそ
ひとりでモテモテに？！
3番電脳倶楽部様お会計です

げっ電クラおまえ・・・・
び・・・美白？！っていうかCD-ROM化？！

カッコいい？楽しいゲームに便利なツール、投稿満載のX68用ディスクマガジンなんだけどさ
中味はそのままか
モテモテは遠いな

本質はかわらなくとも、中味は変わるんですよ、ホントは。（担）

電脳倶楽部には面白いモノが色々載っているよ、といえばそうなんですけど、これらは基本的に読者、ユーザーのみんなが作ったものなのです。これがどういうことかというと、電脳倶楽部はあなたが作った何かを発表できる場であるということ。プログラムやCG、音楽に小説等アイデア次第。そしてメディアもFDからCD-ROMへとクラスチェンジするそうなので、その幅もますます広がろうというものです。どうです？そろそろあなたも一度見てみませんか。そして参加してみませんか。

松本健一（静岡県）

9784797309980
1929455021905
Printed in Japan クニメディア　ISBN4-7973-0998-9 C9455 ¥2190E　雑誌 65814-82

SOFTBANK MOOK
1999夏号
ソフトバンクパブリッシング株式会社
定価（本体2,190円）+税